VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Zpětnovazební učení pro řízení optimalizovaných vestavěných systémů

Reinforcement learning for optimised embedded systems control

2019 Bc. Jakub Novák

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Zadání diplomové práce

Student: **Bc. Jakub Novák**

Studijní program: N2649 Elektrotechnika

Studijní obor: 2612T041 Řídicí a informační systémy

Téma: Zpětnovazební učení pro řízení optimalizovaných vestavěných systémů
Reinforcement Learning for Optimized Embedded Systems Control

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Popis energeticky nezávislých vestavěných zařízení.
2. Teorie metod zpětnovazebního učení.
3. Implementace částí simulátoru optimalizovaných vestavěných systémů v jazyce C#.
4. Implementace vybrané metody zpětnovazebního řízení v simulátoru.
5. Testování implementované metody.
6. Export výsledků simulace a odborná diskuze.
7. Zhodnocení přínosu práce.

Seznam doporučené odborné literatury:

[1] SUTTON, Richard S a Andrew G BARTO. *Reinforcement learning: an introduction.* Cambridge, Mass.: MIT Press, c1998. ISBN 978-0262193986.
[2] MARTIN, Robert C a Micah MARTIN. *Agile principles, patterns, and practices in C#.* Upper Saddle River, NJ: Prentice Hall, c2007. ISBN 978-0131857254.
[3] MUSILEK, P., Michal PRAUZEK, Pavel KROMER, James RODWAY a Tomáš BARTOŇ. Intelligent Energy Management for Environmental Monitoring Systems.(2017) *Smart Sensors Networks: Communication Technologies and Intelligent Applications*, 2017. pp. 67-94. ISBN 978-0-12-809859-2. https://doi.org/10.1016/B978-0-12-809859-2.00005-X.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **doc. Ing. Michal Prauzek, Ph.D.**

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * IDIG * OSTRAVA *

doc. Ing. Jiří Koziorek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 30. dubna 2019

Jakub Novák

Rád bych zde poděkoval panu doc. Ing. Michalu Prauzkovi Ph.D a Ing. Jaromíru Konečnému Ph.D za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování diplomové práce věnovali.

Abstrakt

Cílem této práce je optimalizace využití energie nezávislého vestavěného systému. Pro dlouhodobé měření znečištění ovzduší je potřeba zařízení umístit na odlehlou lokalitu, kde není možné napájet zařízení z eletrizační soustavy. K tomuto účelu byl zvolen řídicí algoritmus z oblasti zpětnovazebního učení, který je schopen naučit se žádaného chování na základě předepsané strategie. V práci jsou popsány principy zpětnovazebního učení a energeticky nezávislých vestavěných systémů. Testování je provedeno v simulačním prostředí a jeho výsledky jsou srovnány s převzatým algoritmem časově závislého řízení. Pro simulaci byly použity meteorologická data v průběhu čtyř let. Vytvořený algoritmus vykazuje větší robustnost, minimalizuje množství energie ztracené v důsledku plného nabití superkapacitoru a zamezuje jakýmkoliv výpadkům energie. Na základě zjištěných údajů je možné algoritmus použít na fyzickém zařízení.

Klíčová slova: Zpětnovazební učení, Q-learning, Energetický nezávislý vestavěný systém, Dependency injection

Abstract

The aim of this thesis is to optimize the energy use of an independent embedded system. For long term measurements of air pollution, it is necessary to place the device in a remote location, where it is not possible to supply energy demands from power grid. For this purpose a reinforcement learning algorithm has been choosen. Which is able to learn the desired behaviour based on the prescribed policy. The thesis describes the principles of reinforcement learning and energy independent embbeded systems. Testing is done in simulation enviroment and its results are compared with time based control. For simulation, meteorogical data was used over the course of four years. The created algorithm exhibits greater robustness, minimizes supercapacitor overcharging and prevents any power outages. Based on the simulation results controlling algorithm is capable of long term control and can be used on target device.

Key Words: Reinforcement learning, Q-learning, Energy independent embedded system, Dependency injection

Obsah

Seznam použitých zkratek a symbolů **8**

Seznam obrázků **9**

Seznam tabulek **10**

1 Úvod **11**

2 Energeticky nezávislé vestavěné zařízení **12**
2.1 Jádro systému . . . 13
2.1.1 Mikroprocesor . . . 13
2.1.2 Mikrokontrolér . . . 14
2.1.3 Hradlové pole . . . 14
2.1.4 Systém na čipu . . . 14
2.2 Energetický zdroj . . . 15
2.3 Úložiště energie . . . 17
2.4 Senzorická část . . . 18
2.4.1 Měření teploty . . . 19
2.4.2 Měření koncentrace plynů . . . 19
2.4.3 Měření záření . . . 20
2.5 DC-DC konvertory . . . 20

3 Strojové učení **21**
3.1 Učení s učitelem . . . 21
3.2 Učení bez učitele . . . 21

4 Zpětnovazební učení **22**
4.1 Historie . . . 22
4.2 Princip . . . 22
4.3 TD učení . . . 24
4.4 Algoritmy zpětnovazebního učení . . . 25
4.4.1 Q-learning . . . 26
4.4.2 SARSA . . . 28
4.4.3 DQN . . . 29
4.5 Průzkum vs. využívání . . . 30
4.5.1 Strategie softmax . . . 31
4.5.2 Strategie horní jistoty . . . 31
4.5.3 Strategie $\epsilon-$greedy . . . 32

4.6 Přidělování odměn 32

5 Dependency injection **33**

5.1 Ukázka implementace návrhového vzoru DI 34

6 Simulační software **37**

7 Implementace části simulačního softwaru **40**

7.1 Logovací systém 40

7.2 Řídicí algoritmus 41

7.3 Nastavená strategie řízení 45

8 Testování algoritmu **46**

8.1 Algoritmus Q-learning 46

8.2 Časově závislé řízení 50

8.3 Testování vybraného algoritmu na bludišti 53

9 Závěr **55**

Literatura **57**

Přílohy **61**

Seznam použitých zkratek a symbolů

S_t	–	Aktuální stav systému
R_t	–	Obdržená odměna
A_t	–	Zvolená akce
$V(S_t)$	–	Hodnotová funkce daného stavu
$Q(s, a)$	–	Kvalitativní hodnota akce v daném stavu
ϵ	–	Konstanta průzkumu
α	–	Konstanta rychlosti učení
γ	–	Konstanta snižování odměny s časem
π	–	Naučená strategie
DI	–	Dependency injection
MPU	–	Mikroprocesor
MCU	–	Mikrokontrolér
FPGA	–	Hradlové pole
SoC	–	Systém na čipu
MPPT	–	Sledování maximálního bodu výkonu
TD	–	Temporal difference
DQN	–	Deep quality network
RL	–	Zpětnovazební učení
RTD	–	Odporové teplotní detektory
NDIR	–	Nedisperzní infračervené senzory
NDVI	–	Normalizovaný index vegetace
MDP	–	Markovův rozhodovací proces

Seznam obrázků

1 Schéma energeticky nezávislého vestavěného systému 12
2 Schéma zapojení napájecího řetězec s MPPT regulátorem 16
3 Schéma NDIR senzoru 19
4 Apogee PAR senzor 20
5 Dělení strojového učení 21
6 Princip zpětnovazebního učení 23
7 Princip DQN 29
8 Princip vstřikování závislostí 33
9 Schéma simulačního softwaru 37
10 Rozdělení stavů systému 43
11 Správa energie algoritmu Q-learning 46
12 Řízení periferií algoritmu Q-learning 47
13 Správa energie časově závislého kontroléru 50
14 Řízení periferií časově závislého kontroléru 51
15 Aplikace Q-learning ve 2D bludišti 54

Seznam tabulek

1 Přehled mikroprocesorů 13
2 Přehled mikrokontrolérů 14
3 Možné zdroje energie 15
4 Přehled dostupných baterií 18
5 Používané metody zpětnovazebního učení 26

1 Úvod

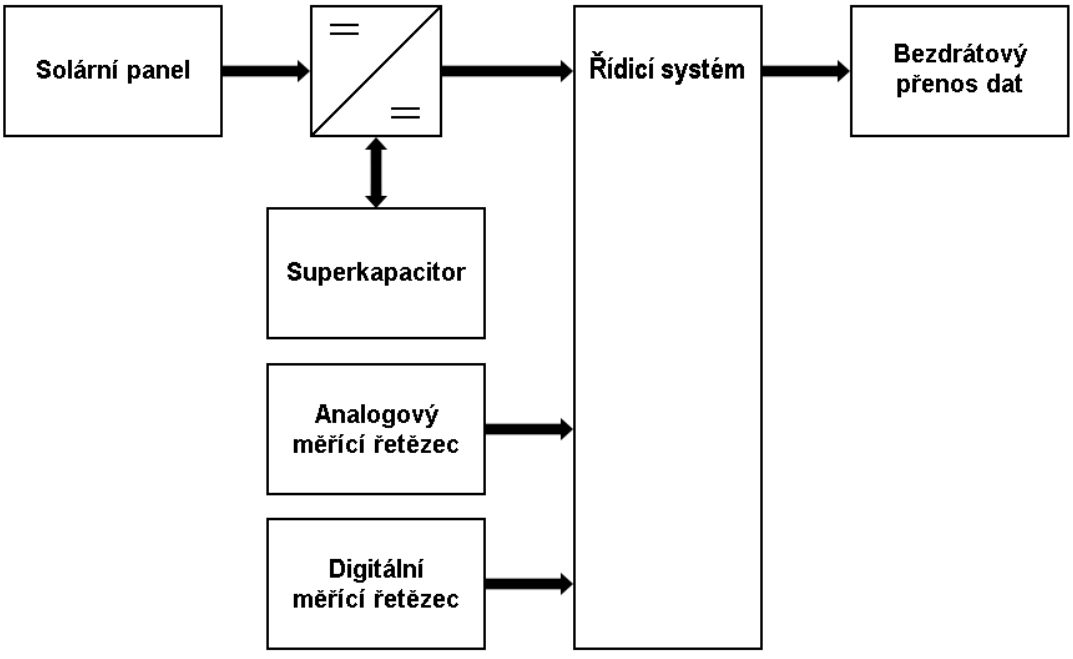
Cílem této práce je vytvořit algoritmus schopný řízení vestavěného systému, který slouží pro monitorování podmínek prostředí. Aby mohly být výsledky prohlášeny za platné, nesmí být ovlivněny přítomností člověka. Z toho důvodu není možné zařízení napájet z elektrizační soustavy. Napájení systému je zajištěno pomocí alternativních zdrojů energie. Je žádoucí, aby systém pořídil co nejvíce vzorků vzhledem k obdržené energii a omezil výpadky měření a energetické ztráty při nabitém energetickém úložišti Vestavěné systémy mají slabší výpočetní výkon a není tedy možné použít komplexní algoritmy pro řešení tohoto problému. Pro správu energie byl použit algoritmus z oblasti zpětnovazebního učení.

Clearing je nejpoužívanější nástroj z této oblasti a to právě proto, že byla prokázána schopnost algoritmu nalézt optimální řešení pro každý konečný Markovův rozhodovací proces. Jeho použití přináší řadu výhod a to zejména schopnost adaptace systému na změnu podmínek. Oproti jiným řídicím systémům je zde značně snížené množství nastavitelných expertních proměnných. Implementovaný algoritmus byl naučen z meteorologických dat obsahujících informace o slunečním svitu v průběhu čtyř let, je tedy schopen předpovídat následující vývoj atmosférických podmínek a dle toho ovládat systém. Chování algoritmu bylo otestováno ve vytvořené simulaci, na jejíž implementaci jsem se podílel. Vytvořil jsem více úrovňový systém pro ukládání výstupních dat a případných chyb. V simulaci jsou použity relativně nové návrhové vzory, které zvyšují přehlednost a odstraňují závislosti mezi jednotlivými bloky. Výsledky adaptivního algoritmu jsou srovnány s převzatým algoritmem časově závislého řízení a to podle stavu energetického úložiště. Po dokončení testování bude systém zasazen do fyzického zařízení.

V práci jsou popsány nejpoužívanější algoritmy zpětnovazebního učení a algoritmy s největším potenciálem rozvoje v následujících letech. Téma jsem si vybral, protože spadá do oblasti strojového učení, což je jedna z nejrychleji se rozvíjejících oblastí současnosti. V průběhu dalších deseti let lze očekávat masivní rozšíření strojového učení přes většinu oborů. Kritickým bodem bude vytvoření víceúčelové umělé inteligence, výzkumní pracovníci z DeepMind tvrdí, že se o to zaslouží právě metoda hlubokého zpětnovazebního učení.

2 Energeticky nezávislé vestavěné zařízení

Vestavěný systém je zařízení, které na jedné desce integruje hardwarové obvody spolu s programem navrženým pro řešení určitého problému. Jedná se o maximálně optimalizovaný systém pro specifickou aplikaci. To zaručuje minimální cenu a velikost. Řídicí jádro systému může být MPU (mikroprocesor), MCU (mikrokontrolér), SoC (systém na čipu), FPGA (hradlové pole). Často se používá pro real-time aplikace. Každý takový systém obsahuje hardwarové komponenty, které mu umožňují činnost a interakci s okolím, jde hlavně o zdroj, řídicí jednotku, paměť, časovače, výstupní obvody, komunikační porty, specifické obvody pro danou aplikaci. Energeticky nezávislý systém znamená, že zařízení nevyžaduje trvalé ani krátkodobé připojení k elektrizační soustavě pro svou činnost. Měřící systémy tohoto typu začínají v této oblasti dominovat. Důvodem je snížení ceny za údržbu energetického úložiště, zároveň systémy monitorující environmentální veličiny budou přítomností člověka ovlivněny. Takový systém může teoreticky fungovat nekonečně dlouho. [21] Aby tento systém mohl fungovat, musí být opatřen zdrojovou částí, ta je obvykle složená ze zařízení, které slouží k přeměně energie na elektrickou energii (solární panel, větrná elektrárna atd.), obvodů pro úpravu elektrické energie například řízení nabíjení baterií a samotného zařízení pro uchovávání elektrické energie. Pro získávání údajů z okolního prostředí je systém opatřen senzorickou částí, ta je tvořena analogovým či digitálním měřícím řetězcem dle použitého senzoru. Sběr dat je prováděn bezdrátově pomocí nízko výkonových technologií jako je LoRa nebo SigFox. Je však možné využít také GPRS moduly. Blokové schéma znázorňující jeho strukturu je znázorněno na obrázku 1.



Obrázek 1: Schéma energeticky nezávislého vestavěného systému

Je důležité, aby zařízení splňovalo energetickou neutralitu, porušení této podmínky může vést k výpadkům celého systému. Podmínka stanoví, že množství spotřebované energie musí být vždy menší nebo rovno energii získané z prostředí. Toto je vyjádřeno rovnicí (1). [13] Cílem optimalizačních algoritmů je dodržení této podmínky a zároveň maximalizace vzorkování.

$$B_0 + \eta \int_0^T [P_s(t) - P_c(t)]dt - \int_0^T [P_c(t) - P_s(t)]dt - \int_0^T P_{leak}dt \geq 0 \quad (1)$$

$P_s(t)$ představuje množství energie získané z prostředí v daném čase, naopak $P_c(t)$ je množství energie spotřebované. Ve vzorci je namodelováno realistické úložiště energie, kde P_{leak} reprezentuje samovolné vybíjení energetického úložiště v čase. Převod energie ze zdroje do úložiště není bezztrátový, tento fakt je namodelován konstantou η. Původní hladina energie energetického úložiště při spuštění systému je zde zavedena přes člen B_0. [13]

2.1 Jádro systému

V dnešní době existuje celá řada řídicích jednotek, ze kterých si může uživatel vybrat. Při výběru platformy hraje hlavní roli instrukční soubor, registrové vybavení, způsob zpracování přerušení a v neposlední řadě zkušenost programátora. Ve většině případů jde o volbu mezi mikrokontrolérem a mikroprocesorem. [27] Jeho úkolem je ovládat periferie systému podle použitého algoritmu.

2.1.1 Mikroprocesor

Na jednom čipu je obsažena výpočetní jednotka s velkou mírou integrace. Skládá se z aritmeticko-logické jednotky (ALU), pole registrů, řadiče a jednotky pro výpočet s plovoucí čárkou. Při své činnosti MPU následuje sekvenci vyzvednutí dat, dekódování a provedení operace. Řízení MPU je prováděno instrukční sadou, která může být buď redukovaná, nebo komplexní. Výkon je dán počtem operací za sekundu (MIPS). [27] Tabulka 1 uvádí základní přehled mikroprocesorů, které je možné použít pro vestavěné aplikace. Na trhu je rozsáhlé množství mikroprocesorů a je jen na uživateli, pro které zařízení se rozhodne.

Tabulka 1: Přehled mikroprocesorů [18]

Mikroprocesor	**Výrobce**	**Architektura**
68HCxx	Motorola	CISC
80x86	Intel	CISC
SPARC	Sun	RISC
ARM	NXP, Atmel, STM	RISC s CISC

2.1.2 Mikrokontrolér

Mikrokontrolér je kompaktní integrovaný obvod navržen pro řízení specifické operace ve vestavěném systému. Na rozdíl od MPU lze MCU používat bez pomocných externích komponent, jelikož všechny potřebné jsou integrovány v obvodu. Jde o obvody A/D - D/A převodníky, komparátory, časovače a komunikační rozhraní. [27] Základní přehled mikrokontrolérů, které se často používají, je uveden v tabulce 2. V dnešní době jsou často používané MCU s 32-bitovou architekturou. Výkonnostně jsou na nižší úrovni, než mikroprocesory. Použití mikrokontroléru jako řídicího zařízení pro tuto úlohu se jeví jako nejvýhodnější, jelikož spotřebu energie lze nejlépe minimalizovat. Použité algoritmy v kapitole 8 pracují s desetinnými čísly, je tedy vhodné vybrat MCU, který obsahuje FPU.

Tabulka 2: Přehled mikrokontrolérů [18]

Mikrokontrolér	**Výrobce**	**Architektura**
68HC11xx,HC12xx,HC16xx	Motorola	CISC
8051,8501MX	Intel, Philips	CISC
PIC 16F84, 16C76,16F876, PIC18	Microchip	CISC
Vylepšení MCU Cortex-M3 ARM9/7	Texas, Philips, Samsung, STM	RISC s CISC

2.1.3 Hradlové pole

Obvody FPGA disponují vysokým výpočetním výkonem. Jsou vysoce flexibilní s možností rekonfigurace. Hradlové pole obsahuje matici konfigurovatelných logických bloků, pomocí nichž je tvořená cílová aplikace. Vstupně - výstupní obvody slouží pro komunikaci s okolními periferiemi. Prvky jsou spolu propojeny pomocí lokální a globální sítě. Na čipu se také nachází statická paměť, násobičky a bloky pro generování hodinového signálu. [17] Hlavní použití těchto obvodů spočívá ve vytvoření hardwarového návrhu. Není vhodné je použít pro výpočet s desetinnou čárkou, jelikož nemají integrovanou FPU.

2.1.4 Systém na čipu

Elektronický integrovaný obvod, který obsahuje potřebné komponenty pro vykonávání určité aplikace. Tato technologie je široce rozšířená a to hlavně díky chytrým telefonům a nositelné elektronice. SoC umožňuje vyrábět menší, výkonnější a energeticky méně náročně zařízení. Na většině těchto čipů se vyskytuje CPU, GPU, RAM, ROM a modem. Největšími výrobci SoC jsou Qualcomm, Samsung a MediaTek. [24] Tato technologie je zde zmíněna, protože patři do skupiny vestavěných zařízení, avšak pro konkrétní aplikaci, kterou práce zpracovává, není tento přístup vhodný. Důvodem je nízká možnost optimalizace spotřeby.

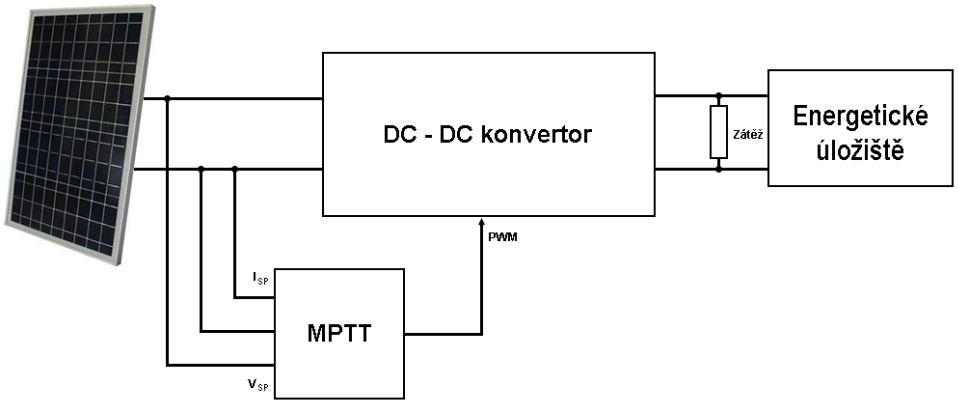
2.2 Energetický zdroj

Existuje celá řada způsobů, jak získat z okolního prostředí energii pro napájení systému. Je však třeba vzít v potaz aplikaci systému a jeho umístění. V tomto případě by bylo nevhodné použít zdroj energie, jehož obsluha, nebo vedlejší produkty by ovlivnili měření systému. Zároveň musí být energie ze zdroje dostatečná na provoz zařízení k němu připojené. Možné zdroje jsou popsány v tabulce 3.

Tabulka 3: Možné zdroje energie [30]

Technologie	**Výkonová hustota**
Solární (vnější)	$15\,\mathrm{mW \cdot cm^{-3}}$
Větrné	$380\,\mu\mathrm{W \cdot cm^{-3}}$
Piezoelektrické	$330\,\mu\mathrm{W \cdot cm^{-3}}$
Vibrační	$116\,\mu\mathrm{W \cdot cm^{-3}}$
Termoelektrické	$40\,\mu\mathrm{W \cdot cm^{-3}}$
Akustické	$960\,\mathrm{nW \cdot cm^{-3}}$

Zde se jeví nejvýhodněji použití solárního panelu jako zdroje. Množství energie vyrobené solárním panelem je řádově vyšší, než u jiného zdroje energie, i přes relativně nízkou účinnost převodu (21 %). Fotovoltalické články přeměňují dopadající elektromagnetickou energii světla na elektrickou a to díky fotovoltalickému jevu. Podle technologie výroby mohou být rozděleny na typy monokrystalické, polykrystalické a amorfní. Nejvyšší účinnost a dodávaný výkon mají články monokrystalické avšak za vyšší cenu. [36] Typické výstupní napětí jednoho článku je půl voltu. Články se skládají do panelů. Výstupní výkon závisí na intenzitě světla, velikosti plochy panelu a dílčích účinnostech převodu energie. Pro maximalizaci získané energie je potřeba mít panel nastaven v optimální pozici. U stacionárních panelů to je pod úhlem 30° - 45° s orientací na jih. Výkon solárních panelů závisí na solární konstantě, která se mění s pozicí. Jde o množství zářivého toku dopadajícího na m^2 zemské atmosféry. Na fyzickém zařízení je realizován jako autonomní fotovoltaický systém, ten se skládá z panelu, regulátoru a akumulátoru. Další výhodou je absence mechanických částí, což snižuje pravděpodobnost vzniku poruchy a potřebný zásah z venčí. Pro maximální extrakci energie ze solárního panelu je dnes běžnou praxí použít MPPT (Maximum power point tracking) 2. Ten pracuje s daty proudu a napětí solárního panelu a podle nich nastavuje PWM signál řídící DC-DC konvertor. Existuje celá řada řídicích algoritmů jednotek MPPT, nejpoužívanější spočívá na principu vyvolání poruchy a pozorování změny. I v této oblasti se začínají uplatňovat algoritmy umělé inteligence jako například fuzzy, neuronové sítě, evoluční algoritmy. [3]



Obrázek 2: Schéma zapojení napájecího řetězec s MPPT regulátorem

Pro použití větrných turbín pro generování energie hraje nejpodstatnější faktor umístění. Kinetická energie větru je pomocí lopatek přenášena na hřídel generátoru, ve kterém se mění mechanická energie na elektrickou. Pro reálné větrné elektrárny lze vypočíst výkon dle vzorce (2).

$$P = c_p \cdot \rho \cdot \frac{v^3}{2} \cdot \pi \cdot \frac{D^2}{4} \tag{2}$$

Součinitel výkonnosti c_p je v ideálním případě roven 0,59. Ze vzorce je patrné, že výkon závisí na rychlosti větru v a průměru rotoru D. Upravením počtu listů rotoru je možné ovlivnit výkon, avšak na úkor účinnosti. Benetzovo pravidlo udává maximální využití energie, to je okolo 59%. [19] Při srovnání se solárními panely mají větrné elektrárny řadu nevýhod, nižší produkci energie, obsahují mechanické části a vyžadují pravidelné kontroly.

Vibrační zdroje fungují na principu přeměny mechanické energie vibrací na elektrickou energii. Této oblasti je v poslední době věnováno spousta pozornosti právě kvůli potenciálu miniaturního zdroje energie. Tyto zdroje využívají elektromagnetickou, elektrostatickou, nebo piezoelektrickou konverzi. Elektromagnetické využívají Faradayův zákon indukce proudu v cívce. Elektrostatické fungují na principu změny vzdálenosti mezi kapacitními deskami při konstantním napětí. Nevýhodou tohoto systému je, že při spuštění vyžaduje externí zdroj. Nejefektivnější je piezoelektrický zdroj. Vlivem tlaku na piezoelektrický materiál dochází k separaci náboje. Pro maximalizaci získané energie je potřeba systém nastavit na dominantní frekvenci prostředí. [33] Do této oblasti spadají i zdroje akustické. Williams a Yates vytvořili vestavěné zařízení schopné sběru energie z okolí. Stanovili, že množství energie je proporcionální třetí mocnině frekvence. Pro takové zařízení je typická hodnota výkonu 0,1 mW při frekvenci 330 Hz. [37]

Termoelektrické zdroje získávají energii z médií o rozdílných teplotách. Účinnost takového systému je dána Carnotovou rovnicí(3).

$$\eta = \frac{T_{teplá} - T_{studená}}{T_{teplá}} \tag{3}$$

Je dána teplotami médií. Při teplotním gradientu 10 °C je účinnost systému 3,3 %. Většina takových systémů je založena na Seebeckově jevu. Výstupní napětí takových systémů je přibližně jeden volt. Nedávno vyvinutý prototyp byl schopen vyrobit 40 µW při teplotním rozdílu 5 °C. [32]

2.3 Úložiště energie

Energetické úložiště přináší do systémů tohoto typu řadu výhod. Výstupní energie ze zdroje může být proměnná, baterie hladinu napětí stabilizují a zajistí, aby energetické požadavky systému byly naplněny. Každá technologie energetických úložišť má unikátní vlastnosti a každé se hodí na jiné aplikace. Pro výběr ideálního úložiště pro danou aplikaci je potřeba se zamyslet nad technickými a ekonomickými kritérii. [20] V energeticky nezávislých systémech je často primární a sekundární úložiště. Primární má malou kapacitu, což mu umožňuje rychlé nabíjení, z ní je napájen celý systém. Sekundární má velkou kapacitu a při dosažení optimálního stavu je zde převáděna energie. To zároveň usnadňuje uvedení systému do provozu. [21]

Energetická a výkonová hustota

Je udávána ve $\mathrm{W \cdot kg^{-1}}$ a je odvozena z poměru množství energie k objemu. Do objemu jsou započtena všechna zařízení systému energetického úložistě.

Provozní omezení

Jsou dána umístěním systému a požadavky. Jde hlavně o teplotu a bezpečnost systému.

Časová odezva

V některých aplikací je potřeba dodat energii do systému do specifického času.

Samovybíjení

Část energie, která byla v původním stavu uložena, se po čase ztratí.

Životnost

Je množství cyklů, které je energetické úložiště schopno zvládnout.

Účinnost

Proces nabíjení a vybíjení může způsobit značné ztráty. Ty jsou přímo vázány na technologii článku.

Cena

Pro určení ceny energetického úložiště je cena definována na nabíjecí cyklus.

Tabulka 4: Přehled dostupných baterií [7] [29]

Typ baterie	**Napětí (V)**	**Výkonová hustota ($\mathbf{Wh \cdot kg^{-1}}$)**	**Teplota (°C)**	**Počet cyklů (−)**
Olověný článek	2,1 - 1,8	30 - 40	-20 až 60	$1 \cdot 10^2 - 5 \cdot 10^2$
Ni-Cd	1,3 - 0,8	40 - 60	-40 až 70	$2 \cdot 10^3$
Ni-MH	1,3 - 0,9	70 - 100	-20 až 40	$1 \cdot 10^3$
Li(Ni/Co/Mn)O_2 - C	4,2 - 2,5	120 - 160	-30 až 45	$3 \cdot 10^2 - 1 \cdot 10^3$
$LiFePO_4$ - C[57]	3,5 - 2,5	80 - 90	-30 až 45	$1,5 \cdot 10^3 - 2 \cdot 10^3$
LiPo	4,0 - 2,4	100 - 110	-20 až 60	$6 \cdot 10^2$
NaS	2,1 - 1,8	60 - 120	245 až 350	$4 \cdot 10^3$
Na(Fe/Ni)Cl	2,6	50 - 10	-5 až 40	$3 \cdot 10^3$
VRB	1,6 - 1,1	10 - 20	10 až 40	$5 \cdot 10^3$
Superkapacitor	2,2 - 3,3	5 - 15	-40 až 65	$1 \cdot 10^5 - 1 \cdot 10^6$

Ve výše vyobrazené tabulce 4 jsou uvedeny baterie dostupné na trhu spolu s jejich parametry. Olověný článek je často používaný díky nízké ceně a vysoké účinnosti při nabíjení, avšak je limitován nízkým počtem cyklů a úzkým intervalem pracovních teplot. Typickým využitím je akumulace elektřiny z fotovoltalických elektráren. Ni−Cd akumulátory mají vysokou životnost, schopnost rychlého nabíjení. Jsou odolné vůči vibracím a nárazům. Mezi nevýhody patří menší měrná energie na hmotnost, na konci vybíjecí charakteristiky je prudký pokles napětí. U tohoto typu článku je častý paměťový efekt. Ni−MH mají oproti Ni−Cd větší kapacitu při stejných rozměrech, nehraje zde roli paměťový efekt a jsou šetrnější k životnímu prostředí, díky absenci kadmia. [7] Naopak mají větší samovybíjení a vnitřní impedanci. Li−Ion má vysokou hustotu energie a nominální napětí, velice malé samovybíjení. Tento článek ztrácí kapacitu bez ohledu na četnost používání, při špatném použití hrozí samovznícením, jak zjistil Samsung. Li−Po články mají podobné parametry jako Li−Ion. Na fyzickém zařízení byl jako úložiště energie použit superkapacitor a to hlavně kvůli vysokému počtu nabíjecích cyklů. Superkapacitor má také vyšší rychlost nabíjení a vyšší spolehlivost. V porovnání s ostatními energetickými úložišti mají superkapacitory mnohem rychlejší samovybíjení, také je jejich cena odhadnuta na pětinásobek olověných článků. Parametry konkrétního úložiště energie byly převedeny do simulace.

2.4 Senzorická část

Jejím cílem je sběr informací o okolních podmínkách. Existuje nesčetné množství senzorů, které je možné použít, za hlavní zmínku stojí senzory teploty, detektory koncentrace plynů a senzory světelného záření. Můžeme je rozdělit na aktivní a pasivní a podle měřícího řetězce na analogové a digitální. Dle použitých senzorů je potřeba upravit jejich zapojení, zejména úroveň napájecího

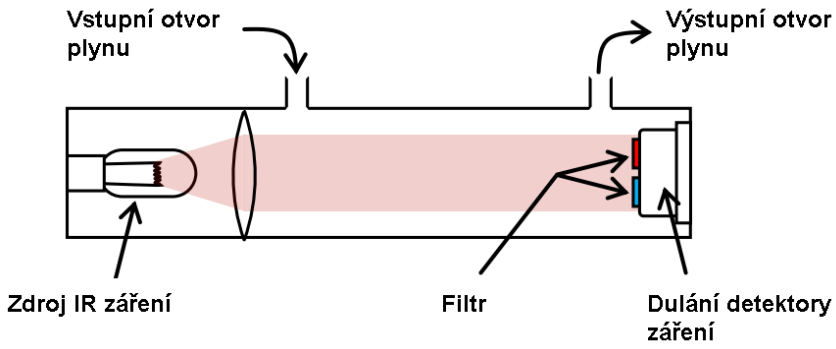
napětí. Odchylka měření je dána součtem dílčích nejistot měřícího řetězce. Celý měřící řetězec je pak připojen do řídicí části systému.

2.4.1 Měření teploty

Teplotní senzory lze rozdělit na odporové a kovové. Nejpoužívanější typy jsou RTD (odporové teplotní senzory), v různých provedeních, polovodičové (PTC, NTC) a termočlánky. Nejčastější materiál pro RTD senzory je platina (PT100). Na trhu je celá řada uspořádání těchto senzorů. Omotávané, odporový vodič je navinut na keramické jádro, celý článek je zapouzdřen. Plošné, na izolační materiál je nanesená vrstva odporového vodiče. Napájením senzoru konstantním proudem lze vypočíst úbytek napětí, ten je přímo vztažen k teplotě. TCR konstanta specifikuje, o kolik se zvýší odpor s přibývající teplotou. Tyto senzory se vyznačují lineární charakteristikou v širokém pásmu, v aplikacích pod 600 °C vytlačují termočlánky.

2.4.2 Měření koncentrace plynů

Jako senzor koncentrace plynů v oblasti analýzy ekosystémů má největší význam detektor CO_2. Ty lze rozdělit na NDIR (infračervené) a chemické. Infračervené pracují na principu absorpce charakteristické vlnové délky záření plynem. Testovaný plyn vstupuje do trubice, ve které je umístěn zdroj záření a detektory, podle Beer-Lambertova zákona je záření absorbováno. Záření je filtrováno dvěma pásmovými propustmi. Aktivním detektorem záření je určena absorpce, ta je porovnána s referenčním detektorem 3. [11] Podle úrovně absorpce je odvozena koncentrace plynu. Nejlepší senzory dokáží detekovat koncentraci až na hranici 20 ppm. Chemické obsahují citlivou vrstvu polymeru. Mají nízkou spotřebu, avšak po delší době se přesnost měření snižuje.



Obrázek 3: Schéma NDIR senzoru [11]

2.4.3 Měření záření

PAR senzor slouží k detekci záření využívané rostlinami pro fotosyntézu. Reaguje na záření na vlnových délkách 400 - 700 nm. Výstupní veličina $\mu mol \cdot m^{-2} \cdot s^{-1}$, neboli zkráceně PPFD specifikuje množství záření na metr čtvereční za sekundu. Senzor se skládá z filtru a detektoru. Filtr působí, jako pásmová propusť na uvedené vlnové délce. Jako detektor je často použita GaAsP fotodioda 4. [8] Detekcí odrazu záření od vegetace je možné změřit kvalitu vegetace. Přítomnost chlorofylu má za následek absorpci světla v červeném spektru. Přítomnost biomasy může být detekována na vlnových délkách blízkých infračerveným. Zdravá vegetace odráží hodně infračerveného světla a málo červené složky spektra, naopak nízká přítomnost chlorofylu bude mít za následek snížení vyzařování ve spektru blízkém infračervenému a zvýšení červeného vyzařování. Tyto dva faktory jsou zkombinovány při výpočtu NDVI. Systémy na měření NDVI jsou umístěny na satelitech (NOAA). [28]

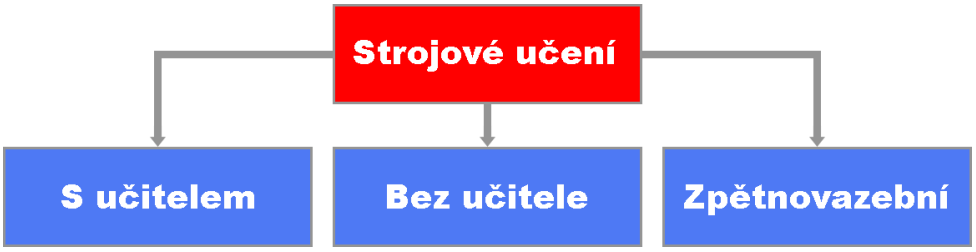
Obrázek 4: Apogee PAR senzor [15]

2.5 DC-DC konvertory

Úroveň napětí ze solárního panelu je nutné snížit na pracovní hladinu řídicího systému. Není vhodné použít lineární regulátory napětí, jelikož dochází k vysokým tepelným ztrátám, účinnost převodu je nízká. Z toho důvodu se používají spínané zdroje, pro tento konkrétní případ je použit „buck“ konvertor. Zde účinnost převodu dosahuje až 90 %. Tranzistor je spínán PWM signálem. Na jeho výstupu je přidán LC filtr, který limituje špičky napětí a proudu. Při rozepnutí tranzistoru se na straně induktoru nahromadí velké množství elektronů, což má za následek negativní napěťovou špičku. Pro její odstranění je paralelně k filtru připojena dioda. Při připojení různých zátěží se projeví úbytek napětí, je potřeba zavést zpětnou vazbu, která bude regulovat střídu řídícího signálu. [31]

3 Strojové učení

Jedná se o typ umělé inteligence, která poskytuje počítačům schopnost se učit bez toho, aby k tomu byly programovány. Strojové učení se zabývá vývojem počítačových programů při vystavení novým datům. V reálném světě existuje řada způsobů jak řešit daný problém, je tomu tak i v oblasti strojového učení. Můžeme ho rozdělit na tři základní kategorie dle obrázku 5.



Obrázek 5: Dělení strojového učení

Učení s učitelem nebo bez učitele potřebují soubor dat. Sestavení souboru dat nemusí být vždy jednoduché, v těchto případech je výhodné použít metodu třetí, tou je zpětnovazební učení.

3.1 Učení s učitelem

Učení s učitelem je podskupina strojového učení, spočívá v tom, že je algoritmu předán soubor dat, který obsahuje vstupní a výstupní data. Z této cvičné množiny je model systému upraven tak, aby co nejlépe aproximoval mapovací funkci. Algoritmus s každou iterací odhaduje hodnoty výstupu, poté je opraven učitelem. Proces učení skončí poté co je učitel spokojen s dosaženou přesností odhadnutých výstupů. Úlohy učení s učitelem mohou být rozděleny do dvou podskupin, regrese a klasifikace. Klasifikace spočívá v určení, do které z oblastí naučeného systému patří nová data. Regrese je odhad funkce z označených dat pro určení budoucích dat. [6] Typickým případem těchto algoritmů jsou neuronové sítě se zpětnou propagací.

3.2 Učení bez učitele

Metoda učení bez učitele je obdoba předchozí metody, s tím rozdílem, že k dispozici jsou pouze vstupní data. Cílem této metody je hledat vzory mezi daty a seskupovat je do klastrů. Umožňuje také odhadnout hustotu dat a redukci dimenzí dat. Tato metoda je výhodná pro hledání struktur ve velkých skupinách dat. Oproti učení s učitelem jsou tyto metody obtížnější na implementaci a to hlavně proto, že má méně testů a modelů, které zajišťují správnost výsledků, celé prostředí je méně kontrolovatelné. Příkladem může být třeba algoritmus k-means, který patří mezi často používané algoritmy hierarchické shlukové analýzy. [6]

4 Zpětnovazební učení

Je třetí oblastí strojového učení. Jako ostatní umožňuje adaptaci na specifický problém. V posledních letech je tato metoda s kombinací neuronových sítí často využívána.

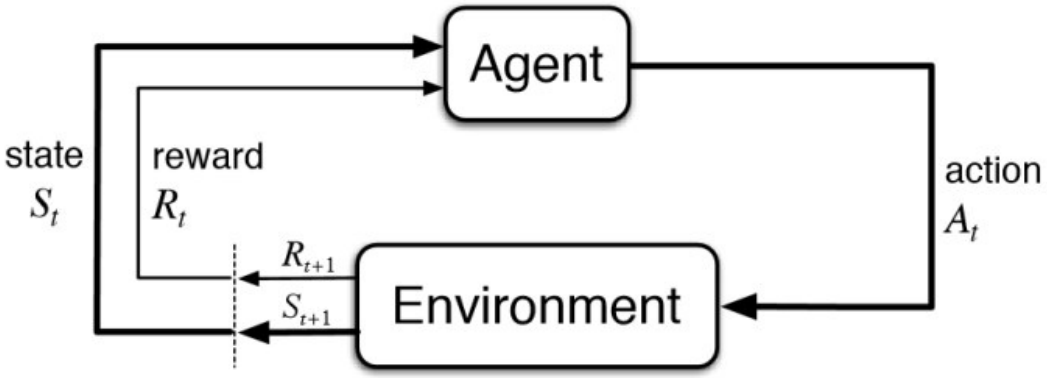
4.1 Historie

Historii lze rozdělit na dvě větve, první se zabývá učením pokus a omyl a druhá se snaží nalézt optimální řízení. První základy principu zpětnovazebního myšlení byly položeny v první polovině minulého století. Psycholog C. Lloyd Morgan, který studoval chování zvířat, zveřejnil metodu učení pokus a omyl. V padesátých letech minulého století byla uvedena Bellmanova rovnice, která umožnila výpočet optimálního řízení známého jako dynamické programování. To vede na rozšíření metod Monte Carlo. O pár let později Bellman zde zavedl také Markovův rozhodovací proces, pojmenovaný po ruském matematikovi Andrey Markov. Ten specifikuje sekvenci událostí, kde každá závisí na předchozí. Poté vývoj zpětnovazebního učení utichá, jelikož se výzkum přesouvá do oblasti učení s učitelem. O návrat k vývoji se postaral H. Klopf, který poukázal na výhody metod pokus-omyl. [38] Na jeho práci staví Richard Stutton a Andrew Barto, kteří se dále podílejí na vývoji zpětnovazebního učení, zejména pak TD učení.

O asi největší průlom v této oblasti se postaral Watkinst (1989), který uvedl nejrozšířenější nástroj oblasti zpětnovazebního učení, Q-learning. Zde dochází ke spojení oblastí metod pokus-omyl a dynamického programování. Další úspěchy zpětnovazební učení dosáhlo teprve nedávno. V listopadu roku 2013 skupina vývojářů z londýnské firmy DeepMind Technologies vytvořila algoritmus, který byl schopen se naučit a hrát sadu her z Atari 2600. K tomu použili kombinaci neuronové sítě a zpětnovazebního učení. O měsíc později byla firma odkoupena firmou Google. V lednu 2016 AlphaGo porazil stávajícího šampiona ve hře Go, o rok později nastoupil Alpha Zero, který předčil všechny předchozí a doposud zůstává nepřekonán. V říjnu tohoto roku byla univerzitou v Berkeley publikována studie o využití zpětnovazebního učení pro replikování složitých pohybů člověka, i zde byl použit algoritmus Q-learning. V budoucnu se počítá i s požitím algoritmu na reálném robotovi od Boston Dynamics. [26] Vědci tvrdí, že zpětnovazební učení je jedním s nejslibnějších nástrojů pro vytvoření univerzálního inteligentního robotického chování. [38]

4.2 Princip

Velká výhoda oproti učení s učitelem nebo učení bez učitele je v tom, že není potřeba cvičných souborů dat. Systém takto natrénovaný z dat poskytnutých expertem se v nejlepším případě bude chovat jako expert sám. Zpětnovazební učení nám však dává možnost překonat schopnosti experta.



Obrázek 6: Princip zpětnovazebního učení [25]

V Markovově rozhodovacím procesu (MDP) je agent, který je zodpovědný za volbu akcí, interaguje s prostředím do kterého je vložený. Tyto interakce se odehrávají následovně s průběhem času. V každém časovém kroku agent vrátí nějakou reprezentaci stavu systému. S ohledem na tuto reprezentaci agent vybere následující akci. Agent musí neustále monitorovat prostředí, protože předchozí akce mohly pozměnit jeho stav. Poté je prostředí přeneseno do dalšího stavu a udělí agentovi odměnu vzhledem k jeho předchozí akci. Přechod mezi stavy, akcemi a odměnami se odehrává sekvenčně a cyklicky. To lze graficky znázornit pomocí tzv. trajektorie. Během procesu má agent za úkol maximalizovat získané odměny obdržené za zvolené akce. Agent chce maximalizovat nejen okamžité hodnoty, ale i kumulativní odměny, které obdrží s postupem času. [34] Získá z toho strategii pro řešení systému, čím je strategie lepší, tím rychleji dosáhne agent cíle. Lze tedy říct, že cílem zpětnovazebního učení je naučení agenta optimální strategii. MDP nám dává možnost formalizovat postupné rozhodování, to je základem pro problémy řešené zpětnovazebním učením. Skládá se z této čtveřice: [12]

- S - konečný počet stavů
- A - konečný počet akcí
- T - pravděpodobnost přechodové funkce

  $$T(s^{'}|s,a) = P(S_{t+1} = s^{'}|S_t = s, A_t = a)$$

- R - odměna

  $$R(s,a) = \mathbb{E}(R_{t+1}|S_t = s, A_t = a)$$

Po ukončení průzkumu všech stavů se agent naučí strategii π. Ta reprezentuje distribuci akcí daných stavů a plně definuje chování agenta (4). Je však nutné dodat, že MDP závisí na aktuálním stavu systému a je časově závislý. [9]

$$\begin{aligned} \pi(a|s) &= \mathbb{P}[A_t = a|S_t = s] \\ A_t &\sim \pi(\cdot|S_t), \forall t > 0 \end{aligned} \tag{4}$$

První rovnice stanoví existenci strategie π při daném stavovém prostoru. Druhá rovnice specifikuje vytvoření strategie π při zvolení série akcí A_t, pro každý čas t větší než nula. Vyřešení problému znamená nalezení strategie, která se blíží optimální. Pro každý případ bude existovat strategie π, která předčí ostatní. [38] Celý proces zobrazen na obrázku 6 je prováděn opakovaně, dokud není dosaženo požadovaného počtu epizod, nebo dokud není dosaženo optimálního řízení. Na začátku agent vybere akci, ta je zvolena náhodně, jelikož je Q-tabulka inicializována na nulové hodnoty. Akce je provedena a ohodnocena systémem, informace o důsledku akce je předána agentovi v podobě odměny, ten podle ní aktualizuje hodnotu v Q-tabulce. Po provedení určitého počtu akcí, agent vybírá akce s nejvyšší hodnotou pro maximalizaci odměny, poté se cyklus opakuje. Rozměry Q-tabulky jsou dány počtem stavů systému a počtem možných akcí, její velikost tedy odpovídá velikosti stavového prostoru. Existuje řada algoritmů, jak agent vybírá nejoptimálnější akci, mezi nejpoužívanější patří Q-learning a SARSA. Tyto a další algoritmy jsou popsány v následující kapitole.

4.3 TD učení

Metod TD učení lze použít pro odhadnutí hodnotové funkce, pokud by nebyl proveden odhad funkce, agent by musel čekat do obdržení finální odměny a poté vysledovat své kroky a upravit hodnoty akcí v daných stavech. [38] Oproti dynamickému programování tato metoda nepotřebuje model systému.

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_t - V(S_t)\Big] \tag{5}$$

Metody TD učení nám umožňují odhadnout finální odměnu v každém stavu a můžeme upravit hodnoty Q-tabulky pro každý krok cyklu.

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[r_{t+1} + \gamma V(s_{t+1}) - V(S_t)\Big] \tag{6}$$

Ve vzorci (6) je nejpodstatnější mnohočlen v závorce. To představuje tzv. „TD error“, jde o rozdíl odhadu maximální hodnoty pro aktuální a budoucí stav. Výsledkem je hodnotová funkce pro daný stav $V(S_t)$. Jsou zde přítomny expertní konstanty (α, γ) ovlivňující učení. Z této funkce je odvozeno aktualizační pravidlo algoritmu Q-learning. Metoda TD je nazývána „bootstraping“, jelikož hodnota akcí v daných stavech jsou aktualizovány částečně pomocí existující odhadované odměny a ne pomocí konečné odměny. [38]

ON strategie TD učení spočívá v naučení hodnoty dané strategie, která je používaná pro výběr akcí. Hodnotové funkce jsou aktualizovány pomocí výsledků ze spuštěných akcí pro danou strategii. Hodnoty akcí v jednotlivých stavech jsou aktualizovány výhradně na zkušenostech. Jedná se většinou o takzvané „měkké“ strategie, to znamená, že je u nich vždy určitý element průzkumu systému. [39] Typickým příkladem pro ON strategii je algoritmus SARSA.

Metody OFF strategie se můžou naučit různé strategie pro chování a odhad. Strategie agenta je také většinou měkká. Zásadní rozdíl oproti ON strategii je v aktualizaci hodnot. Ta může být provedena pomocí zvolení hypotetických akcí, které ještě nebyly vyzkoušeny. To znamená, že OFF strategie může rozdělit průzkum od řízení, ON strategie tohoto není schopna. Agent naučený pomocí OFF strategie může během fáze řízení prokázat vlastnosti, které nebyly patrné ve fázi učení. [39] Příkladem těchto strategií je Q-learning.

Pokud je algoritmus opatřen takzvaným parametrem způsobilosti, hovoříme o TD(λ). Metody TD(λ) se liší od standardních metod TD tak, že aktualizace kvalitativních hodnot je prováděna každých n kroků, nikoliv každý krok. Kdy velikost n je dána parametrem λ. Ke každému páru stav, akce je svázána hodnota způsobilosti. Existují dva způsoby, jak implementovat parametr způsobilosti. U kumulativní cesty jsou parametry inkrementovány po jedničce. V případě nahrazení cesty je parametr způsobilosti daného páru nastaven na hodnotu jedna. [35] [38]

$$e_t(s,a) = \begin{cases} 1 & \text{Nahrazení cesty} \\ \gamma\delta e_{t-1}(s,a) + 1 & \text{Akumulativní cesta} \end{cases} \tag{7}$$

4.4 Algoritmy zpětnovazebního učení

Existuje velké množství algoritmů zpětnovazebního učení, jejich použití záleží na řešeném problému a omezeních při jeho zpracování 5. Většina přístupů spočívá v použití TD učení, jsou však i také co využívají dynamické programování. [38] Metody zpětnovazebního učení jsou definovány strategií, distribucí odměn, hodnotovou funkcí a v některých případech i modelem systému, ve kterém se agent pohybuje. Strategie specifikuje žádoucí chování agenta, je dána sadou pravidel. Ty mohou být sepsány v podobě tabulky, nebo funkce. Je navržená pomocí odměn, tu agent obdrží za akci v každém ze stavů systému. Hodnotová funkce odhaduje množství potenciální odměny, která bude obdržena v delším horizontu událostí. Hodnotová funkce spolu s odměnou determinuje výši hodnoty akce v daném stavu. Tyto algoritmy mohou mít i model systému, to je však výjimečné.

Tabulka 5: Používané metody zpětnovazebního učení

Algoritmus	**Popis**	**Strategie**	**Stavový prostor**
Monte Carlo	Všechny průchody Monte Carlo	OFF	Diskrétní
Q-learning (λ)	Stav-Akce-Odměna-Stav	OFF	Diskrétní
SARSA (λ)	Stav-Akce-Odměna-Stav-Akce	ON	Diskrétní
DQN	Q-learning s využitím DNN	OFF	Spojitý
DDPG	Gradientní determinování strategie	OFF	Spojitý
AC3	Asynchronně výhodný herec-kritik	OFF	Spojitý
NAF	Q-learning s normalizovanou funkcí	OFF	Spojitý
TRPO	Gradientní sestup optimalizace strategie	ON	Spojitý

4.4.1 Q-learning

Jeden z nejpoužívanějších algoritmů zpětnovazebního učení a to právě proto, že byla prokázána schopnost algoritmu nalézt optimální řešení pro každý konečný Markovův rozhodovací proces. Jako paměť systému je použita Q tabulka, ve které jsou uloženy kvalitativní hodnoty jednotlivých akcí. Rozměry tabulky jsou dány počtem stavů a možných akcí. [40] Celý algoritmus je řízen podle rovnice (8).

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \cdot \Big[R_{t+1} + \gamma \cdot \underset{a}{max} Q(S_{t+1}, a) - Q(S_t, A_t)\Big] \tag{8}$$

$Q(S_t, A_t)$ představuje hodnotu v Q-tabulce pro aktuálně zpracovávanou akci A_t a stav S_t. Člen α udává, do jaké míry budou staré informace přepsány novými. Může nabývat hodnot v intervalu ($0 \leq \alpha \leq 1$). V případě, že míra učení bude rovna 0, agent se nebude učit nové informace a bude výhradně používat předchozí poznatky. Míra učení 1 naopak znamená, že agent vezme v potaz nově nabité informace. V praxi je tato konstanta často inicializována na vysokou hodnotu, po nalezení optimálního řízení systému by se měla α rovnat nule, to znamená zastavení učení. Konstanta R_{t+1} představuje míru odměny pro zvolenou akci. Například agent obdrží odměnu, pokud dosáhne cílového stavu, nebo požadovaného chování. Index γ reprezentuje, jak moc budoucí události ztratí svou hodnotu podle toho, jak daleko v budoucnosti se vyskytují, i tento parametr je na intervalu $\langle 0, 1\rangle$. Když je $\gamma = 1$ odměna obdržená nyní má stejný význam jako odměna obdržená v budoucnosti. V opačném případě $\gamma = 0$ agent bude upřednostňovat akce, které povedou k okamžité odměně. To může značně ovlivnit rychlost učení. [38] [1] Nastavení expertních konstant záleží na nastavené strategii, velikosti stavového prostoru a cílovém chování systému. Je běžné nastavit tyto parametry dynamicky. V závorce je uveden rozdíl takzvaného cíle (target) a ceny (cost). Cíl reprezentuje nejvyšší budoucí hodnotu, je dán součinem diskontního faktoru γ a maximální hodnoty z budoucího stavu $\underset{a}{max} Q(S_{t+1}, a)$, k němu je přičtena odměna obdržená systémem R_{t+1}. Člen ceny je dán aktuálním stavem a provedenou akcí $Q(S_t, A_t)$. Jde

o algoritmus s OFF strategií, nepotřebuje model a jeho akce jsou diskrétní. Tento algoritmus má větší rozptyl hodnot než SARSA a může mít problémy s konvergencí, což je častý problém při použití Q-learning pro trénování neuronových sítí. Q-learning je vhodné použít pro učení simulací, nebo tam, kde při vyzvednutí záporné odměny nedojde k ztrátám. [10]

Algoritmus 1 Q-learning [38]

Inicializace $Q(s, a)$, pro všechny $s \in S$, $a \in A(s)$
Inicializace expertních konstant
for pro každý cyklus **do**
  Inicializace S
  while S není cílový stav **do**
    Výběr A z daného S pomocí strategie odvozené z Q(např. $\epsilon - greedy$)
    Použij A a pozoruj R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha\Big[R + \gamma \underset{a}{max} Q(S', a) - Q(S, A)\Big]$
    $S \leftarrow S'$
  end while
end for

V kapitole 4.2 už byla zmíněna struktura kroků pro popisovaný algoritmus. Ty jsou opět patrné ve výše uvedeném algoritmu 1. V počátku se inicializuje paměť a expertní konstanty. Poté se prochází stanovený počet epizod. Při každém začátku epizody dojde k inicializaci počátečního stavu systému. Následuje cyklus, který je prováděn až do splnění cílového stavu. V každém kroku je akce vybrána a aplikována. Systém na ni reaguje a vrací odměnu. Podle zmíněného aktualizačního pravidla (8) je upravena kvalitativní hodnota. Budoucí stav je předán do další iterace. Po ukončení určitého počtu epizod je agent schopen systém úspěšně řídit. Jejich počet závisí na složitosti systému a expertních konstantách.

Existuje také upravená metoda $Q(\lambda)$ je rozšířena o parametr způsobilosti, ten je svázán s každou dvojicí (stav, akce) ve formě paměti. Konstanta λ reprezentuje počet zpětných kroků pro Q^{π}. Parametr způsobilosti s každým krokem klesá s funkcí $\gamma\lambda$. Pokud se strategie upřednostňuje odměny v blízkém časovém intervalu, algoritmus pokračuje v učení, v momentě výběru průzkumné akce se hodnoty způsobilosti pro všechny páry akcí a stavů vynulují. [10] A učení je zastaveno. Nejdelší zpětný rozvoj je poté popsán rovnicí (9).

$$r_{t+1} + \gamma r_{t+2} + \cdots + \gamma^{n-1} r_{t+n} + \gamma^{n} \underset{a}{max} Q_t(s_{t+n}, a) \tag{9}$$

Rozvoj je dán jednotlivými odměnami v budoucích krocích r_{t+n} a konstantou ztráty odměny γ. V posledním kroce je do něj přičtena maximální hodnota akce daného stavu $\underset{a}{max} Q_t(s_{t+n}, a)$.

4.4.2 SARSA

Tento algoritmus je velice podobný Q-learning. Hlavním rozdílem mezi nimi je, že u SARSA algoritmu maximální odměna nemusí být vždy použita pro aktualizaci hodnot v Q-tabulce 10. Místo toho je zvolena nová akce a dopočtena odměna za použití stejných zásad, které určily původní akci. Název algoritmu vychází z toho, že se aktualizace provádějí pomocí pětice $Q(s, a, r, a^{'}, s^{'})$. Kde a a s určují původní akci a stav, r je odměna obdržení za následující stav a $a^{'}$ a $s^{'}$ je budoucí akce a stav. Algoritmus spadá do kategorie s ON strategií a diskrétními akcemi. Stejně jako Q-learning i tento přístup nevyžaduje model systému. [38]

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \cdot \Big[R_{t+1} + \gamma \cdot Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big] \tag{10}$$

Aktualizační pravidlo tohoto algoritmu je velice podobné pravidlu algoritmu Q-learning. Jediný rozdíl spočívá ve formulaci cíle, ten je založen na budoucí vybrané akci. Ostatní symboly rovnice (10) jsou stejné jako u předchozího algoritmu, proto zde nejsou vysvětleny. Q-learning se jednoznačně naučí optimální strategii, zatímco SARSA se při průzkumu naučí strategii, která se bude blížit optimální. Pro dosažení optimální strategie pomocí tohoto algoritmu je potřeba vhodně zvolit strategii výběru akce a postupně snižovat parametr ϵ. Přesné nastavení může být obtížné. Díky možnosti penalizace průzkumných akcí je SARSA více konzervativnější než Q-learning. Pokud je při průzkumu velké riziko záporné odměny blízko optimální cesty Q-learning tuto odměnu spustí, zatímco SARSA bude mít tendenci se ji vyhnout. Pomalu se tuto cestu naučí používat při snižování parametru průzkumu. V praxi je SARSA použita v případě, kdy se nejedná o simulaci a chyba v učení může stát peníze. [10]

Algoritmus 2 SARSA [38]

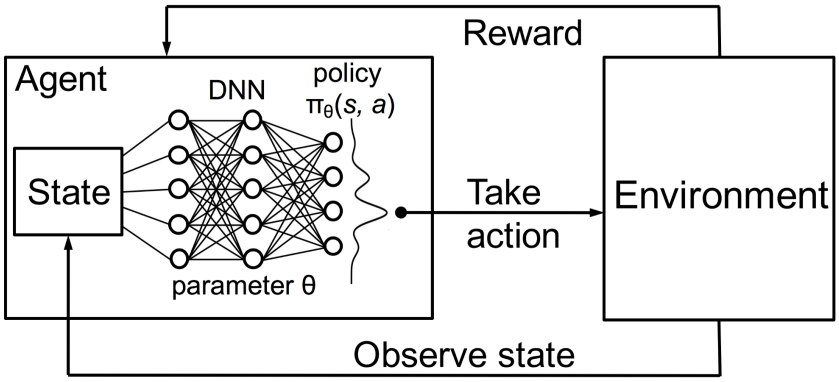
Inicializace $Q(s, a)$, pro všechny $s \in S$, $a \in A(s)$
Inicializace expertních konstant
for pro každý cyklus **do**
  Inicializace S
  Vyber A z daného S pomocí strategie odvozené z Q(např. $\epsilon - greedy$)
  while S není cílový stav **do**
    Použij A a pozoruj R, $S^{'}$
    Výběr $A^{'}$ z $S^{'}$ pomocí strategie odvozené z Q(např. $\epsilon - greedy$)
    $Q(S, A) \leftarrow Q(S, A) + \alpha\Big[R + \gamma Q(S^{'}, A^{'}) - Q(S, A)\Big]$
    $S \leftarrow S^{'}$; $A \leftarrow A^{'}$
  end while
end for

Zásadní rozdíl oproti algoritmu Q-learning spočívá ve výběru akce, ta je vybrána ještě před započetím epizody. Uvnitř každé epizody se opět akce aplikuje. Poté je zde vybrána budoucí

akce z budoucího stavu, ta bude hrát roli v aktualizačním pravidlu. Do další iterace je předán jak budoucí stav, tak budoucí akce.

4.4.3 DQN

I přes vysokou efektivitu algoritmu Q-learning mu chybí jistá všeobecnost. Agent u Q-learning nemá jak vybrat z akcí, ve stavu, který ještě nebyl objeven. Tento problém byl upraven algoritmem DQN (Deep Quality Network). Tento přístup se používá hlavně u aplikací, u kterých je stavový prostor spojit7, je potřeba použít aproximační nástroje (neuronové sítě) pro určení hodnoty stavu. Systém se skládá ze dvou různých neuronových sítí (cílová a hodnotící) se stejnou strukturou, s jinými váhami a parametry. Klíčová myšlenka spočívá v gradientním sestupu dle algoritmu (3) 7. Kde y_j je cílová hodnota a $Q(\phi_j, a_j; \theta)$ je odhad, cílem je minimalizovat rozdíl mezi nimi. Po určitý počet kroků je cílová hodnota konstantní, poté dochází k přepsání parametru θ hodnotou stejnojmenné proměnné z cílové neuronové sítě, tím je zajištěno učení. [1] Tento přístup je jedním z mála algoritmů, které umožňují naučení se více systémů. Tuto vlastnost prokázali ve firmě DeepMind, naučením agenta více her, ve kterých si vedl lépe než jakýkoliv člověk. Zatím je však omezen počtem systémů, které je schopen se naučit. [14] Zpětnovazební učení není nějak zvlášť náročné na výpočetní výkon, v předešlých letech se prokázalo jako užitečný a efektivní nástroj. S každým rokem se zvyšuje výkonost počítačů a proto se oblasti umělé inteligence začínají prosazovat výpočetně náročné nástroje, které v minulých letech nenašly velké využití. Také proto dochází ke kombinaci Q-learningu a těchto nástrojů.



Obrázek 7: Princip DQN [22]

Algoritmus 3 DQN [23]

Inicializace paměti D na kapacitu N
Inicializace hodnotové funkce Q náhodnými váhami
for cyklus $= 1, M$ **do**
  Inicializace sekvence $s_1 = x_1$ a předběžně zpracované sekvence $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    if $\epsilon >$ náhodné číslo **then**
      Výběr $a_t = max_a Q^*(\phi(s_t), a; \theta)$
    else
      Výběr náhodné akce a_t
    end if
    Nastavení $s_{t+1} = s_t, a_t, x_{t+1}$ a preprocesu $\phi_{t+1} = \phi(s_{t+1})$
    Uložení přechodu $(\phi_t, a_t, r_t, \phi_{t+1})$ do paměti D
    Vzorkuj náhodný počet přechodů $(\phi_j, a_j, r_j, \phi_{j+1})$ z paměti D
    Nastavení $y_j \begin{cases} r_j & \text{pro koncový } \phi_{j+1} \\ r_j + \gamma max_{a'} Q(\phi_j, a'; \theta) & \text{pro nekoncový } \theta_{j+1} \end{cases}$
    Proveď gradientní sestup na $(y_j - Q(\phi_j, a_j; \theta))^2$ podle rovnice:
    $$\bigtriangledown_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a \sim p(\cdot); s' \sim \mathcal{E}} \left[\left(r + \gamma max_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i) \right) \bigtriangledown_{\theta_i} Q(s, a; \theta_i) \right]$$
  end for
end for

4.5 Průzkum vs. využívání

Ve zpětnovazebním učení vzniká konflikt mezi výběrem optimální akce a prozkoumáváním systému. Problém nastává pří nalezení optimálních akcí. Při vysokém poměru průzkumu je zaručeno nalezení globálního minima, avšak po naučení optimální strategie je průzkum nežádoucí, jelikož by vedl k volbě vedlejších akcí. Naopak při volbě upřednostňování odměny nemusí být nalezeny potencionální odměny. Tato expertní konstanta je volena v závislosti na rozsahu stavového prostoru a nastavené strategii. Při rozsáhlém stavovém prostoru je dobré konstantu nastavit na průzkum, aby bylo zajištěno objevení všech stavů. [5] V kódu výběr akce probíhá vygenerováním náhodného čísla a následovným porovnáním s konstantou. Jak už bylo zmíněno, toto rozhodnutí závisí na definované konstantě ϵ. Pokud je $\epsilon = 1$ systém bude vybírat maximální hodnoty z tabulky. Agent se dostane do cílového stavu systému, je však pravděpodobné, že to nebude optimální cestou. Může se také jednat pouze o lokální minimum. Nastavením konstanty $\epsilon = 0$ bude agent čistě prozkoumávat systém bez ohledu na získané odměny. Vhodným zvolením konstanty je potřeba nastavit chování agenta (např. $\epsilon = 0.9$). V 90 procentech bude vybrána akce s maximální hodnotou, ve zbylých 10 pak bude vybrána náhodně. Avšak i po několika cyklech, kdy se agent plně naučí prostředí, se může stát, že zvolí náhodné akce a nedostane se do cíle op-

timální cestou. Existuje celá řada metod, jak tento problém vyřešit. Asi nejjednodušším řešením by bylo vynásobit konstantu ϵ exponenciální funkcí v závislosti na cyklu dle rovnice (11).

$$e^{-(1-\epsilon)\cdot iterace} \tag{11}$$

Tímto zajistíme klesání konstanty ϵ. Po uplynutí 23 cyklů bude šance na výběr náhodné akce pouhých 10 procent. S dalšími iteracemi se bude $\epsilon \to 0$. Nyní si popíšeme nejpoužívanější strategie pro odstranění problému zmíněného výše.

4.5.1 Strategie softmax

Tento přístup zapojuje výběr akce v závislosti na vážené pravděpodobností. Největší výhodou oproti $\epsilon - greedy$ strategii je, že agent může vzít v potaz pravděpodobnosti pro výběr ostatních akcí. Při použití Boltzmannovy rovnice průzkumu jsou akce řazeny podle jejich relativní hodnoty. Tímto způsobem může agent ignorovat akce, které byly odhadnuty jako neoptimální a věnovat se těm, které jsou do budoucna slibné (12). [16]

$$P_t(a) = \frac{exp(q_t(a)/\tau)}{\sum_{i=1}^{n} exp(q_t(i)/\tau)} \tag{12}$$

Základní předpoklad je, že tato metoda poskytuje informaci o jistotě agenta u vybrané akce. Agent odhaduje měřítko toho, jak optimálně si myslí, že je akce zvolená. V praxi se používá dodatečný parametr „teplota“ τ, který se snižuje s časem. Tento parametr řídí šíření softmaxové distribuce. Všechny akce jsou na začátku učení rovnocenné, na konci jsou pak řídce distribuovány. Parametr $q_t(a)$ odpovídá očekávané odměně následující akce. [38]

4.5.2 Strategie horní jistoty

Spolu s odhadováním výsledku akce nám tato strategie předepsaná rovnicí (13) umožní pracovat s jistotou, se kterou byl odhad učiněn. Tato strategie výběru akce je založená na velice optimistickém přístupu. Spočívá ve výběru akce, která vede k nejlepšímu možnému výsledku, ač je tento výsledek jakkoliv nepravděpodobný. Výběr akce je prováděn podle rovnice uvedené níže.

$$A_t \doteq argmax\left[Q_t(a) + c\sqrt{\frac{lnt}{N_t(a)}}\right] \tag{13}$$

Konstanta c určuje míru průzkumu systému. Parametr $N_t(a)$ symbolizuje, kolikrát byla daná akce vybrána ve vztahu k času. S postupem času budou vybrány všechny akce. Akce, jejichž aplikací se nedosáhlo výrazného postupu, nebo akce často volené budou se stárnutím agenta vybírány s menší četností. Implementace tohoto postupu je náročnější, než $\epsilon - greedy$ a pro většinu případů řešených pomocí zpětnovazebního učení nepraktický. Algoritmus je zejména nevhodný v případech, kde je velké množství stavů. [5] [38]

4.5.3 Strategie $\epsilon-$greedy

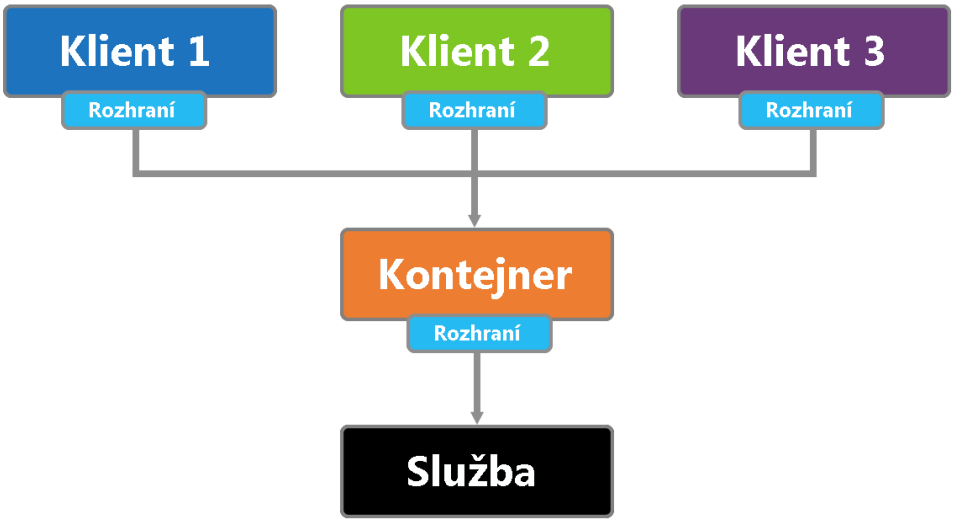
Jde o asi nejjednodušší algoritmus, jak vyřešit problém průzkumu a využívání. Princip spočívá v zavedení pravděpodobnosti, se kterou bude vybrána optimální nebo náhodná akce. Pravděpodobnost zvolení optimální akce je pak $1 - \epsilon$, naopak akce náhodná má pravděpodobnost ϵ. V jednoduchých případech je ϵ konstantní. Obvykle se však snižuje s postupem času tak, že strategie používaná asymptoticky se stane chamtivou. [38] Neboli optimální $\hat{\pi} \rightarrow \pi^*$. Toho může být dosaženo právě když $\epsilon_k = \frac{1}{k}$ kde $k \rightarrow \infty$. Z rovnice je patrné, že ϵ_k bude konvergovat k nule. Pravděpodobně nejpoužívanější algoritmus použit firmou DeepMind při vytváření umělé inteligence na hraní her Atari 2600. [23] Nedá se přesně určit, která ze zmíněných strategií výběru je nejlepší. Schopnost učení podle jednotlivých metod bude záležet na daném problému.

4.6 Přidělování odměn

Přidělování odměn je asi nejobtížnější částí implementace zpětnovazebního učení, i přesto, že při navrhování funkce nás nesvazují žádná pravidla. Rozdělování odměn záleží na daném problému, nelze proto navrhnout univerzální přidělování odměn. Záleží také, zda je stavový prostor časově spojitý, či diskrétní. Například v časově závislém problému není dobré přidělovat odměny vzhledem k vzdálenosti aktuálního stavu od cílového. Takové případy je možné vyřešit pomocí hierarchického RL. Odměny je potřeba navrhnout tak, aby směřovaly ke chtěné strategii. Je třeba dát pozor na to, aby konvergence byla co nejrychlejší a agent nezůstal zaseknutý v lokálním minimu. Velice často jsou kladné odměny přiděleny v dosažení cílového stavu a záporné s každým dalším krokem agenta, přičemž výše odměny je úměrná důležitosti dosažení stavu. [4] Tím je zajištěna rychlá konvergence. Pro případ použití kombinace neuronové sítě s RL je potřeba normalizovat hodnoty v Q-tabulce.

5 Dependency injection

Jde o metodu programování podle daného návrhového vzoru. Závislost mezi jednotlivými objekty vzniká tehdy, pokud je pro činnost objektu vyžadována činnost jiného objektu. Například třída vyžaduje data z databáze. DI v podstatě funguje tak, že závislost mezi objekty je z venku vtlačena do vytvářeného objektu. A to pomocí setteru nebo jako v našem případě pomocí konstruktoru. Oddělí se tak konstrukce třídy od konstrukce jejich závislostí. Závislostně inverzní princip říká, že kód by měl záviset na abstrakci. [2] Tímto zajistíme oddělení jednotlivých implementací objektů 8. V *C#* pro abstrakci použijeme rozhraní, tímto způsobem můžeme zaměňovat objekty, pokud splňují požadované rozhraní, což zajistí vysokou modularitu. O sestavení a inicializaci závislostí se stará kontejner, ten provede vytvoření instance v momentě vyžádání první třídy. Ulehčuje tak práci programátorovi, který nemusí závislosti zajišťovat. Je potřeba předat všechny potřebné parametry. Použitím DI oddělíme kód od implementace na nižší úrovni. Kód bude tak přehlednější a bude mnohem lehčí ho modifikovat. Návrhový model zajistí také vysokou soudržnost a nízkou spřaženost, zamezíme tím tak šíření případných chyb mezi jednotlivými moduly.



Obrázek 8: Princip vstřikování závislostí

5.1 Ukázka implementace návrhového vzoru DI

Jak už bylo zmíněno, tento návrhový vzor slouží k odstranění závislostí mezi jednotlivými třídami. Jeho použití je při práci s takto rozsáhlými systémy velice výhodné. Pro demonstraci principu tohoto návrhového vzoru je způsob implementace popsán na jednom z bloků simulačního softwaru a to na konkrétní třídě EnergySource. Rozhraní, které bylo použito v simulačním softwaru a na kterém bude vysvětlena funkčnost, je uvedeno ve výpise 1. Definicí rozhraní specifikujeme metody, které objekt musí implementovat, tím je zobecněn použitý prvek. Jsou zde vypsány pouze deklarace funkcí. I v případě, že v následující třídě, která bude z tohoto rozhraní dědit, nebude potřeba nějaké z uvedených metod, musí v ní být tato metoda definována.

```
1    public interface IComponent : IDisposable
2    {
3       void CommandLineSetup();
4       void InitConfig();
5       void Init();
6    }
7
8    public interface IEnergySource : IComponent
9    {
10      double DoIteration(double[] inputData);
11   }
```

Výpis 1: implementace rozhraní

Rozhraní EnergySource implementuje další metody z rozhraní IComponent, kde jsou deklarovány metody pro inicializaci, převod dat z příkazového řádku a inicializaci konfigurace objektu, Interface IDisposable definuje metodu pro uvolnění zdrojů. Takto jsou rozhraní provedena pro všechny třídy simulačního softwaru. Teď stačí zavést konkrétní třídu EnergySource, v tomto případě to může být solární panel nebo větrná elektrárna. Ve třídě musí být implementovány všechny metody definované v rozhraní. Přes konstruktor třídy jsou předány instance na ostatní třídy, které budou potřeba pro správnou funkčnost třídy Solar.

```
1   public class Solar : IEnergySource
2   {
3       public Solar(ISimulationParameters simulationParameters,
4                    ILogger logger,
5                    IDataLogger datalogger)
6       {
7           this.simulationParameters = simulationParameters;
8           this.logger = logger;
9           this.datalogger = datalogger;
10      }
11
12      private readonly ISimulationParameters simulationParameters;
13      private readonly ILogger logger;
14      private readonly IDataLogger datalogger;
15
16      public void CommandLineSetup()
17      {
18      }
19
20      public void Dispose()
21      {
22      }
23
24      public double DoIteration(double[] inputData)
25      {
26      }
27
28      public void Init()
29      {
30      }
31
32      public void InitConfig()
33      {
34      }
35  }
```

Výpis 2: implementace třídy

Takto bude vypadat předpis třídy Solar, výše uvedený kód slouží jako demonstrace řešení DI, kód v uvedených metodách zde není podstatný. Odkazy na globální třídy jsou předány privátním instančním proměnným a je možné zde použít metody i těchto tříd bez vytváření závislostí. Hlavní metoda DoIteration slouží pro specifikování chování konkrétní komponenty, v tomto případě je v komponentě Solar uveden algoritmus pro výpočet energie podle vstupních dat a parametrů solárního panelu. Parametry jsou načteny v metodě InitConfig přes datový soubor.

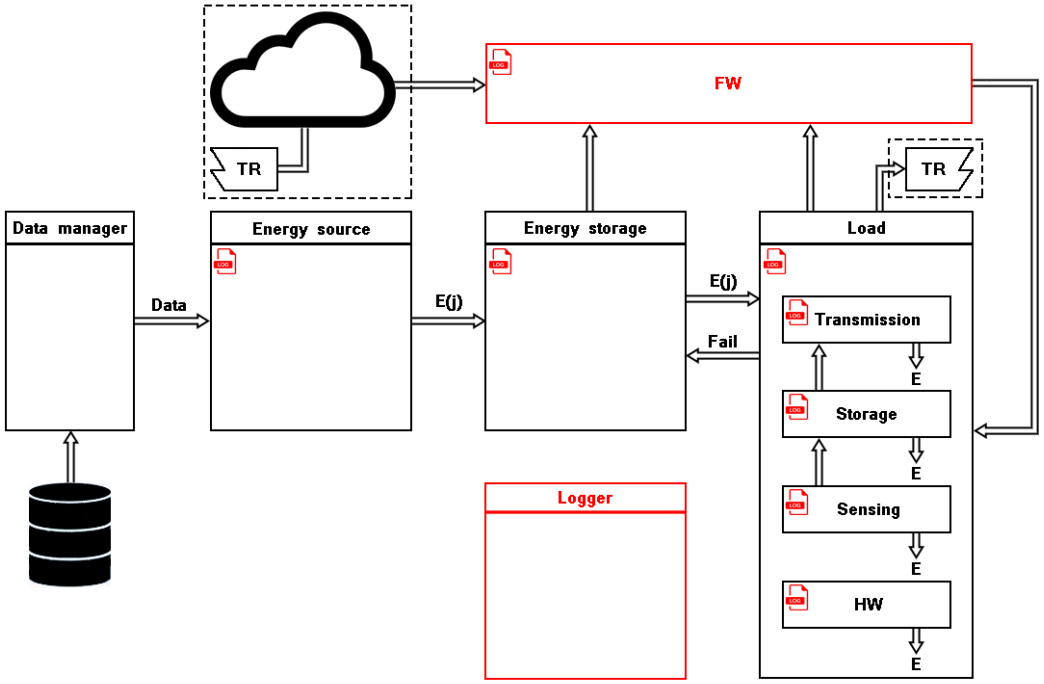
```
1    public void ConfigureServices(IServiceCollection services)
2    {
3       services.AddSingleton<IEnergySource, Solar>(); // Energy source
4    }
```

Výpis 3: registrace služby v kontejneru

Objekt je potřeba zaregistrovat do kontejneru služeb, tím je zároveň specifikována životnost služby. V simulačním softwaru jsou všechny objekty registrovány jako globální, to znamená, že v celém softwaru existuje pouze jedna instance dané třídy. Objekt zaregistrovaný jako Singleton se vytvoří při prvním vyžádání této služby, to je patrné z výpisu 3. Pro každý další požadavek je použita tato již vytvořená třída.

6 Simulační software

Převzatý program jsem doplnil o mnou vytvořené třídy, umožňují činnost celého softwaru. Tyto komponenty jsou ve schématu 9 vyznačeny červeně.



Obrázek 9: Schéma simulačního softwaru

Před spuštěním simulace je potřeba zadat parametry simulace v podobě argumentů. Jejich tvorba a funkčnost je popsána v následujících kapitolách. Simulační software je napsán v jazyku *C#* .NET Core, skládá se z bloků, které definují činnost simulace. V simulačním softwaru jsou použity moderní návrhové vzory, které zvyšují přehlednost a snižují spřaženost celého systému. Jednotlivé bloky jsou spolu provázány přes rozhraní, což činí systém modulární. Do bloku lze dosadit libovolnou třídu, reprezentující chování fyzického prvku. Jelikož jsou od sebe jednotlivé třídy odděleny, jejich výměna nebude mít žádný vliv na okolní třídy, pouze na chování systému. Aby bylo možné komponentu do softwaru vložit, musí implementovat všechny metody předepsané rozhraním. Tento způsob propojení je popsán v kapitole 5. Data jsou extrahovány a distribuovány z datového souboru pomocí bloku Data manager, je zde implementován také kalendář. Datový manager rovněž provádí kontrolu konzistence dat. Data jsou předána bloku Energy source, kde je implementován konkrétní zdroj, v případě solárního panelu jsou zde uvedeny jeho parametry a vypočtena vyrobená energie vzhledem k slunečnímu záření ze vstupních dat. Úložiště energie získává energii od zdroje a distribuuje ji zátěži. Ta pomocí getteru získá spotřebu energie jednotlivých tříd v ní umístěných. Je proveden jejich součet a celková spotřeba jednoho cyklu simulace je vrácena. Poté je proveden rozdíl spotřebované energie s energií v úlo-

žišti podle rovnice (14), pokud je množství energie nedostatečné, dochází k selhání. Všechny nedostatky energie jsou zaznamenány do výstupního souboru simulátoru. V takovém případě systém nepřejde do další iterace, bude proveden Rollback. Při dostatku energie bude proveden Commit a přechod do další iterace. Akce rollback a commit slouží pro potvrzování akce nastavené kontrolérem.

$$\text{Fail} = f(E_{cap}, E_{load}) = \begin{cases} 0 & E_{cap} \geq \sum E_{load} \\ 1 & \text{jinak} \end{cases} \tag{14}$$

Každý z vyobrazených bloků dědí základní metody z třídy Component. Jde o metody Init, InitConfig, CommandLineSetup a Dispose. Funkce Init slouží pro základní inicializaci například vytvoření a otevírání souboru, zápis do záznamového systému. Metoda InitConfiguration slouží k inicializaci konfigurace daného objektu, jde o deserializaci json, základní nastavení těchto parametrů a jejich distribuci do ostatních metod v třídě. Ve funkci CommandLineSetup je zaveden překlad argumentů z příkazového řádku, metoda Dispose slouží k ukončovacím akcím, například uzavírání souborů a odpojování služeb. Při simulaci zde byl vložen superkapacitor, ten plní roli energetického úložiště, jeho simulační vlastnosti odpovídají vlastnostem fyzického. V simulaci plní roli energetického úložiště solární panel, jeho fyzikální funkčnost je zde implementována. Jejich technické parametry jsou vneseny do simulace a jsou použity při výpočtu energie. Příkladem parametrů pro solární panel je jeho plocha, účinnost, umístění. U supekrapacitoru se jedná o parametry maximální energie, pracovní teploty, životnost, efektivita nabíjení, samovolné vybíjení a další. Do většiny bloků se načítají konfigurační parametry konkrétních prvků ze souboru json. Parametry jsou extrahovány pomocí externího nástroje FluentCommandLineParser. Ten je na začátku inicializován a je mu přiřazen argument, který má očekávat. Při zachycení argumentu je při inicializaci simulace objekt deserializován a parametry jsou uloženy do třídy, přes kterou se k nim přistupuje. V bloku zátěže je proveden součet požadavků energií od zapnutých periferií. Jsou zde implementovány metody pro potvrzení a vrácení nastavení periferií, v jednotlivých komponentách uvnitř zátěže jsou definovány výše spotřebované energie v průběhu jednoho cyklu. Nastavení periferií lze provést zápisem do proměnných. Jde o povolení hardwaru, vysílání, měření a datového úložiště.

```
1    controllerActionParameters.DataStorageEnabled = true;
2    controllerActionParameters.HardwareEnabled = true;
3    controllerActionParameters.SensingEnabled = true;
4    controllerActionParameters.TransmissionEnabled = true;
```

Výpis 4: Ukázka nastavení akce (Všechny periferie zapnuty)

Na výše zobrazeném výpisu je znázorněno zvolení akce AllOn. Ne všechny kombinace jednotlivých proměnných jsou realizovatelné, je potřeba vybrat jen ty, které dávají fyzikální smysl. Například aplikací akce, která vypne hardware, ztrácí řídicí algoritmus kontrolu nad systémem. Pro vypnutí dané periferie stačí upravit zápis na false. Zapnutí jakékoliv z periferií ovlivní celko-

vou spotřebu systému. Nejvíce energeticky náročné je snímání spotřebuje 2 J při každé iteraci. V porovnání s ním vysílání odčerpá 30 mJ, hardware 37 mJ datové úložiště 8 mJ. V bloku FW je implementován kontrolér těchto akcí, nastavuje zmíněné proměnné podle informace o stavu energie. V rámci práce zde byl implementován algoritmus zpětnovazebního řízení.

Pro posouzení, jak byl navržený algoritmus úspěšný, je v simulátoru vytvořen časově závislý kontrolér, se kterým jsou výsledky adaptivního algoritmu porovnány. Na fyzickém zařízení bude umístěno nízkoenergetické bezdrátové vysílání dat (LoRa, SigFox). Data budou zpracovávány v cloudovém úložišti. V rámci simulace byly některé bloky nahrazeny jejich jednoduchými kopiemi „Simple“, jedná se o bloky, jejichž funkce nijak neovlivní simulační výsledky, proto není nutné implementovat jejich rozšířené funkce. Například blok transmission. Není potřeba data nikam posílat, jelikož jsou zpracovávány lokálně. Blok Logger je přes konstruktor vtlačen do ostatních komponent. Jeho cílem je zaznamenávat výstupní a informační data simulačního softwaru, jeho činnost je podrobně popsána v následující kapitole 7. Výstupní data ze simulace jsou zaznamenávána do csv souboru unifikovaně, to usnadní překlad dat v zobrazovacím nástroji. Všechny komponenty, které chce uživatel v simulaci použít, musí být registrovány v konfiguraci pro daný způsob řízení. Při spuštění dochází k inicializaci překladového nástroje příkazového řádku, zároveň jsou zpracovány argumenty. Následuje inicializace konfigurace a služeb. Do logovacího systému je zapsána použitá konfigurace a celý systém je odstartován zavoláním metody Start. Po jejím ukončení jsou služby rozpuštěny metodou Dispose. V softwaru je vytvořená vizualizace běžící na TCP portu 8 000, na tomto webu je vyobrazen aktuální stav systému a vývoj energie v průběhu simulace. V sestavovacím souboru simulace je prováděna smyčka po dobu dostupnosti simulačních dat. Poté následuje vyzvednutí nových dat. Do datového souboru je vložen časový záznam zpracování dat, simulační čas a zpracovávaná data. Dalším krokem je provedení iterace pro energetický zdroj, tím je získána vyrobená energie, ta je opět zapsána do výstupního souboru. Tato energie je přes proměnnou třídy roznesena do většiny komponent. Je provedena iterace pro kontrolér, získáme tak požadovanou energii a nastavení periferií. Opět je proveden záznam nastavení periferií. Další metodou je proveden odečet energie z energetického úložiště. Po provedení kontroly množství zbývající energie je posunut simulační čas o stanovený krok, datový soubor je nastaven na nový řádek a cyklus se opakuje.

7 Implementace části simulačního softwaru

7.1 Logovací systém

Součástí této práce je vytvoření záznamového systému, který bude pořizovat informace o konfiguraci a stavu systému. Je rozdělen na dvě části, simulační a datový loger. Záznamový systém vkládá do textového souboru informace o konfiguraci a možné chyby. Pomocí přetěžování funkcí umožňuje záznam na více úrovních (Info, Debug, Error, Warning, Exception). Druhý logovací systém je určen pro ukládání hodnot stavů systém, jako například stav baterie, požadované energie na provedení akce a stav periferií. Tyto proměnné se ukládají do sloupců pomocí funkce AddData, pro ukončení řádku je potřeba zavolat funkci NewLine.

```
6.3.2019 14:05:17 Warning Using QLearningController
6.3.2019 14:05:17 Debug State of energy storage: 750
6.3.2019 14:05:17 Error Sample error message
6.3.2019 14:05:17 Exception System.ArgumentException:
Selected wrong simulation
Parameter name: QLearningConfiguration
at QLearning Controller.Init() in QLearning.cs:line 308
```

Zprávy vyobrazené výše jsou pouze pro demonstraci funkčnosti logovacího systému simulace, vyobrazené chyby se při simulaci nevyskytují. Soubory jsou k dispozici ve formátu csv. Po provedení profilování časového vytížení procesu byly zjištěny vysoké časové ztráty při zápisu výstupních dat do souboru, provedenou optimalizací byl čas průběhu simulace zkrácen na pouhých 8 vteřin. Simulační soubor obsahuje velké množství záznamů, je potřeba, aby soubor nebyl načítán po jednotlivých záznamech, nýbrž otevřen na začátku simulace a uzavřen na konci. Při spouštění simulace je potřeba zadat cesty na konfigurační soubory jednotlivých logerů (.json). V těchto souborech jsou parametry nastavení logerů, například kam se má soubor uložit, delimiter, výchozí formáty. V případě, že některý z potřebných konfiguračních parametrů není v souboru uveden, jsou v třídě specifikovány výchozí formáty, které jsou pak použity. To minimalizuje vznik chyby ze strany uživatele a zároveň unifikuje strukturu záznamů. Číselné záznamy jsou unifikovány na americký standard, tedy desetinnou tečku. Při vložení desetinné čárky by vznikla chyba v překladu dat do softwaru pro jejich zpracování. Jako oddělovač záznamů byla použita čárka, je však možné ho libovolně změnit. Při prvním spuštění simulace jsou na cestách specifikovaných v konfiguraci vytvořeny soubory, do kterých jsou data vkládána. Pokud jsou soubory již vytvořeny, jsou při dalším spuštění pouze otevřeny. Stávající záznamy jsou ponechány a v zápisu se navazuje na konci souboru. Při selhání logovacího systému uživatel ztrácí informace o simulaci. Aby k tomu nedocházelo, je většina metod ošetřena odchytáváním výjimek. V datovém souboru jsou uložena data z průběhu simulace, ty obsahují informace o si-

mulačním čase, iteraci, dnu, vstupních datech, produkované energii, požadované energii, stavech periferií a zbylé energii v úložišti. Ukázka výstupních dat pro pár záznamů je na následujícím výpisu.

```
2008-01-01 00:01:00,1,0.0006944,9.31,1.00,0.08,0.04,0,1,0,0,1,747.41
2008-01-01 00:02:00,2,0.0013888,9.31,1.00,0.08,0.04,0,1,0,0,1,747.43
2008-01-01 00:03:00,3,0.0020833,9.31,1.00,0.08,0.04,0,1,0,0,1,747.44
2008-01-01 00:04:00,4,0.0027777,9.31,1.00,0.08,0.04,0,1,0,0,1,747.46
```

7.2 Řídicí algoritmus

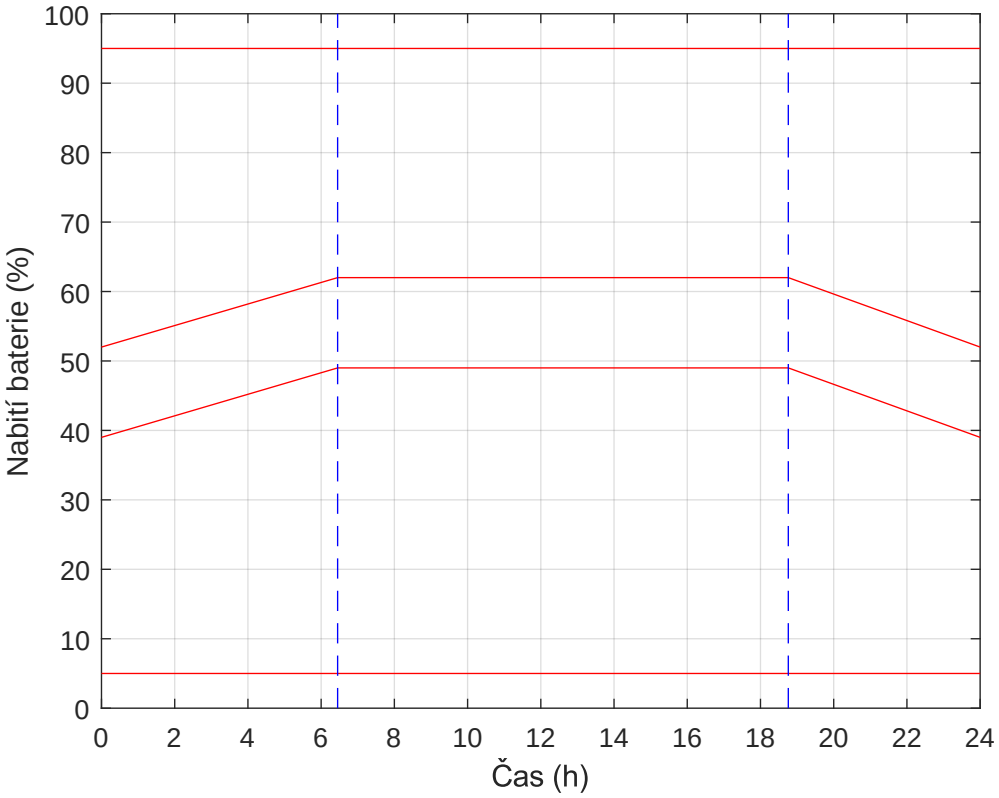
Pro testování chování byl zvolen algoritmus Q-learning. Jde o nejrozšířenější algoritmus z oblasti zpětnovazebního učení. Na začátku jsou přes konstruktor vtlačeny všechny třídy, na jejichž funkčnosti kontrolér závisí. Děje se tak pomocí rozhraní dle návrhového vzoru DI. Jedná se o třídy logovacího systému, simulačních parametrů a vstupních parametrů kontroléru. Při spuštění simulace je Q-tabulka v podobě dvourozměrného statického pole inicializována na nulu. Zde je výhodné, že velikost stavového prostoru je pevně dána a předem známa. V opačném případě by bylo potřeba pole inicializovat dynamicky. Nastavením expertních konstant lze částečně ovládat chování algoritmu při řízení a učení. Jak napovídá jejich název, jejich nastavení záleží na expertovi. Pro optimální nastavení je potřeba znát strukturu a vliv jednotlivých akcí na systém. Konstanta průzkumu ϵ byla v době simulace nastavena na 0,99. Vysoké nastavení redukuje volbu špatné akce v krajních stavech. Jak již bylo zmíněno, kvůli malému stavovému prostoru není třeba rozsáhlého průzkumu. Rychlost učení ovlivňována konstantou α byla nastavena na hodnotu 0,68. To znamená, že výsledný kvalitativní výsledek v paměti bude z 32 % záviset na předešlých hodnotách. Degradace hodnot v paměti byla nastavena na 0,97, to znamená, že algoritmus dává přednost okamžitým odměnám oproti kumulativní odměně. Tento parametr spolu s konstantou učení umožní systému rychlou adaptaci na změnu podmínek. Při takto nastavených hodnotách byly výsledky simulace nejlepší.

Pomocí enumerátorů byly specifikovány všechny možné stavy (Empty, Low, Medium, High, Full) a akce (HwOn, AllOn). Před vkročením do smyčky programu je potřeba inicializovat stav systému. Konfigurační parametry simulace specifikují počáteční stav energetického úložiště jako nabitý, proměnná stavu systému je tedy inicializována na hodnotu Full. Obdobně jsou nastaveny i proměnné pro předchozí stav a akci. Po vstupu do hlavní funkce DoIteration je z kalendáře vyčten čas a převeden na desetinné číslo, podle něj se bude odvíjet přiřazení do stavů. Pro odvození stavu je potřeba vyčíst aktuální stav energie v superkapacitoru přes zavedený parametr kontroléru. Následné zpracování tohoto údaje stanoví, o který z výše uvedených stavů se jedná, kroky zpracování byly následující. Podle údajů z databáze meteorologického ústavu jsem dopočetl průměrnou dobu západu a východu slunce. Pro východ to byl čas 6:27 a pro západ 18:46. Zakřivení funkcí je přizpůsobeno dennímu cyklu slunce a výrazně ovlivňuje řízení

systému algoritmem. Hodnoty energie byly zvoleny dle rozvahy, při pozdějším testování byla snížena hranice stavů Medium-High, tím se dosáhlo nižších energetických ztrát v kapacitoru a pořízení většího počtu vzorků. Dopočtením rovnic přímky byly získány jednotlivé rozsahy stavů systému. Pro určení aktuálního stavu systému je jednoduše pomocí větvení porovnávána energie baterie s vytvořeným modelem stavů, jehož podmínku tvoří časový interval lineární funkce. Rovnice přímek ve vzorci (15) jsou vypočteny pro superkapacitor použitý v simulaci (750 J). Parametr y odpovídá proměnné StateOfEnergy zavedené v programu. Při potřebě provedení simulace s kapacitorem o jiné maximální hodnotě energie stačí přepsat konfigurační parametr energetického úložiště, není potřeba zasahovat do vnitřních funkcí algoritmu. Byla vytvořena funkce, která automatizuje výpočet přímek, jsou jí předány pouze procentuální hranice. Tato funkce vrací konstantní parametry lineární funkce na zadaném intervalu.

$$
\begin{aligned}
\text{Full} \quad &\in \quad y \geq 712,5 \\
\text{High} \quad &\in \left\{ \begin{array}{l} y \geq 11,62t + 390; t \in (0; 6,45\rangle \\ y \geq 465; t \in (6,45; 18,76\rangle \\ y \geq -14,31t + 733,5; t \in (18,76; 24\rangle \end{array} \right\} \wedge \quad y < 712,5 \\
\text{Medium} \in &\left\{ \begin{array}{l} y < 11,62t + 390; t \in (0; 6,45\rangle \\ y < 465; t \in (6,45; 18,76\rangle \\ y < -14,31t + 733,5; t \in (18,76; 24\rangle \end{array} \right\} \wedge \left\{ \begin{array}{l} y \geq 11,62t + 292,5; t \in (0; 6,45\rangle \\ y \geq 367,5; t \in (6,45; 18,76\rangle \\ y \geq -14,31t + 636; t \in (18,76; 24\rangle \end{array} \right\} \\
\text{Low} \quad &\in \left\{ \begin{array}{l} y < 11,62t + 292,5; t \in (0; 6,45\rangle \\ y < 367,5; t \in (6,45; 18,76\rangle \\ y < -14,31t + 636.0; t \in (18,76; 24\rangle \end{array} \right\} \wedge \quad y \geq 37,5 \\
\text{Empty} \quad &\in \quad y < 37,5
\end{aligned}
\tag{15}
$$

Aktuální nastavení rozhraní v úseku s konstantní úsečkou je pro stav Full 100 - 95 %, High 95 - 62 %, Medium 62 - 39 %, Low 39 - 5 % a Empty 5 - 0 %. Grafické zpracování výše uvedených rovnic je vyobrazeno na obrázku 10. Rozdělení stavového prostoru je třeba provést právě kvůli použitému algoritmu Q-learning, ten pracuje pouze s diskrétními stavy. Teoreticky je možné stavový prostor rozdělit na nekonečný počet stavů. Takový prostor je možné považovat za spojitý. V takovém případě je použití tohoto algoritmu nepřípustné. A bylo by potřeba použít jiný z algoritmů zpětnovazebního učení např. DQN, DDPG. Ty jsou však výrazněji náročné na výpočetní výkon a jejich umístění na mikrokontrolér proto není vhodné.



Obrázek 10: Rozdělení stavů systému

Funkce ChooseAction se stará o výběr optimální následující akce. Je jí předán stav systému a Q-tabulka. Podle toho, jestli je náhodná proměnná větší než parametr průzkumu ϵ je vybrána buď akce s maximální hodnotou kvality v daném stavu, nebo náhodná akce. Při vstoupení sytému do předem neprozkoumaného stavu jsou všechny jeho akce nulové, v takovém případě je akce zvolena náhodně. Vybraná akce je vrácena. Následuje funkce EnviromentFeedback, ta vrátí odměnu a budoucí stav pro zvolenou akci. V první fázi je akce provedena pomocí nastavení zavedených parametrů pro řízení periferií, poté je pozorována odezva akce na stav energie. Ta je porovnávána s energií v předchozím kroku. Je potřeba myslet na to, že důsledky akce nastavené v této funkci se projeví až v další iteraci. Podle toho jestli stav energie klesá nebo stoupá, jsou v následující fázi distribuovány odměny dle nastavené strategie popsané v kapitole 4.6. Strategie byla zvolena dle sady pravidel, které definovaly požadované chování řízení. Výše odměn byla nastavena dle uvážení. Ve fázi testování docházelo k přebytečným ztrátám energie v důsledku nabitého energetického úložiště, po optimalizaci velikostí odměn byla tato energie minimalizována. V poslední funkci LearningProcess jsou extrahovány všechny akce z budoucího stavu předaného parametrem, z nich je vybrána maximální akce. Aktualizace hodnoty v Q-tabulce se vkládá do aktuálního stavu a provedené akce. Je proveden součet dosavadní hodnoty na daných indexech se vzorem předepsaným pro algoritmus Q-learning (8). I zde je třeba myslet na aktualizaci hodnot až při dalším spuštění hlavní smyčky. Pak už jsou jen předány aktuální hodnoty energie, stavu

a akce do pomocných proměnných pro další krok algoritmu.

```
1    //zjištění času simulace
2    double time = simulationParameters.Calendar.Hour
3                + ((double) simulationParameters.Calendar.Minute / 60);
4    //zjisti stav systému Full/High/Medium/Low/Empty
5    double energyState = controllerInputParameters.StateOfEnergyStorage[0];
6    //výběr vhodné akce
7    Actions action = ChooseAction(previousState, qTable);
8    (nextState, reward) = EnvFeedback(preAction, time, preEnergy, prevState);
9    //aktualizace hodnot v q-tabulce
10   LearningProcess(previousState, reward, preAction, nextState, qTable);
11   //předání hodnot do další iterace
12   preAction = action;
13   previousEnergy = energyState;
14   previousState = nextState;
```

Výpis 5: Sekvenční zpracování akcí algoritmu Q-learning

Algoritmus implementovaný v simulačním prostředí se nepatrně liší od algoritmu popsaného v kapitole Q-learning 4.4.1 a to hlavně tím, že po nalezení cíle, nebo selhání nedochází k resetování prostředí. Učení probíhá pouze v jedné epizodě. Druhý rozdíl vychází z toho, že provedená akce se projeví až po skončení hlavní smyčky DoIteration. Je proto potřeba v algoritmu pracovat s akcemi a stavy předchozího cyklu. Použitý algoritmus je popsán v následujícím výpise (4).

Algoritmus 4 Použitý algoritmus

Inicializace $Q(s, a)$, pro všechny $s \in S$, $a \in A(s)$
Inicializace expertních konstant
Inicializace S
while Dostupné simulační data **do**
  Výběr A z daného $S_{(t-1)}$ pomocí strategie odvozené z Q (např. $\epsilon - greedy$)
  Použij $A_{(t-1)}$ a pozoruj R, S
  $Q\left(S_{(t-1)}, A_{(t-1)}\right) \leftarrow Q\left(S_{(t-1)}, A_{(t-1)}\right) + \alpha\left[R + \gamma \cdot \underset{a}{max} Q(S, a) - Q(S_{(t-1)}, A_{(t-1)})\right]$
  $S_{(t-1)} \leftarrow S$; $A_{(t-1)} \leftarrow A$
end while

Jako u běžného Q-learning algoritmu se v počáteční fázi nastaví paměťová část, expertní konstanty a počáteční stav, poté jsou procházena simulační data, na které algoritmus reaguje podle kroků popsaných v kapitole 4.4.1. Tedy výběr akce, provedení a aktualizace hodnot v paměti podle předepsaného pravidla.

7.3 Nastavená strategie řízení

V tomto případě byl systém rozdělen do pěti stavů a to podle stavu nabití baterie (Nabito, Vysoká úroveň, střední úroveň, nízká úroveň a vybito). Odměny a penalizace byly přidělovány podle přechodu mezi jednotlivými stavy a podle množství energie. Zda energie roste či klesá, bylo zjišťováno rozdílem aktuálního a předchozího množství energie. V případě dosažení stavu vybití a dalšímu poklesu energie je agent penalizován záporně, při růstu energie je odměněn kladně. Při dosažení stavu plného nabití a klesající energie je odměněn vysokou kladnou hodnotou, při stoupání energie obdrží zápornou. Odměny jsou stejným způsobem nastaveny i u ostatních stavů, liší se pouze výší odměny. Ta byla experimentálně nastavena pro minimalizaci energetických ztrát a výpadků energie. Dále jsou negativně ohodnoceny přechody mimo žádaný stav, tedy Medium. Celý systém je regulován do středního stavu, při přechodu do stavů vedlejších je systém pomocí odměn tlačen zpět. Tato strategie dosahuje žádaného chování a blíží se optimální strategii. Takto distribuované odměny umožnily úspěšné regulování systému, její výsledky jsou popsány v kapitole 8.1. V této podobě je strategie celkem nepřehledná, avšak funkční. Změnou formulace či upravení hodnotové funkce můžeme dosáhnout lepších výsledků při simulaci. Oba okrajové stavy jsou ošetřeny kritickou penalizací, tím se zabrání vybití systému a minimalizují se energetické ztráty. Distribuce odměn by měla být co nejjednodušší na formulaci a zároveň dostatečně popisná, aby plně definovala žádané chování systému.

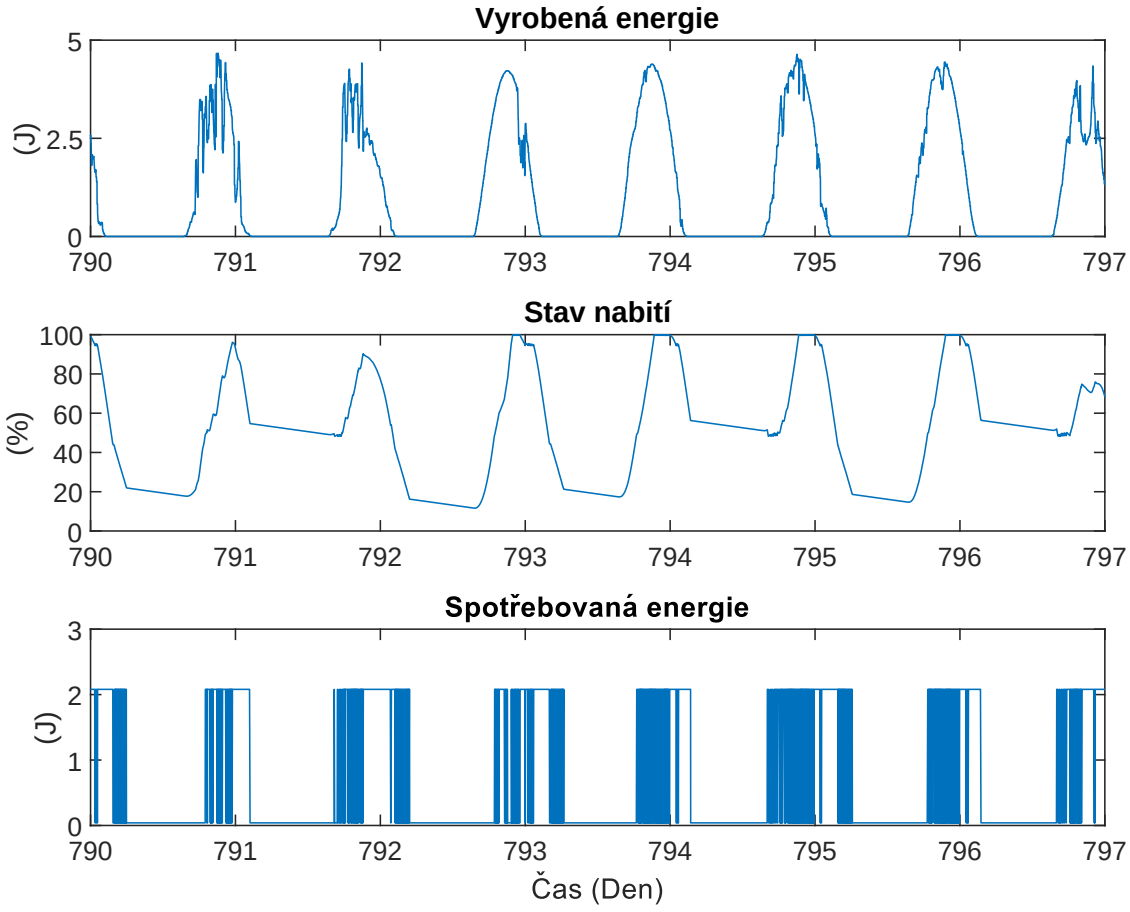
$$R(S,A) = \begin{cases} -1 & S \text{ je Empty} \wedge StateOfEnergy \downarrow \\ -0,75 & S \text{ je Low} \wedge StateOfEnergy \downarrow \\ -0,25 & \text{Medium} \rightarrow \text{High} \cup \text{Medium} \rightarrow \text{Low} \\ 0,5 & \text{Medium} \rightarrow \text{Medium} \wedge StateOfEnergy \downarrow \\ 0,67 & S \text{ je High} \wedge StateOfEnergy \downarrow \\ 1,19 & S \text{ je Full} \wedge StateOfEnergy \downarrow \\ 1 & S \text{ je Empty} \wedge StateOfEnergy \uparrow \\ 0,75 & S \text{ je Low} \wedge StateOfEnergy \uparrow \\ 0,20 & \text{Medium} \rightarrow \text{Medium} \wedge StateOfEnergy \uparrow \\ -0,75 & S \text{ je High} \wedge StateOfEnergy \uparrow \\ -0,78 & S \text{ je Full} \wedge StateOfEnergy \uparrow \end{cases} \tag{16}$$

8 Testování algoritmu

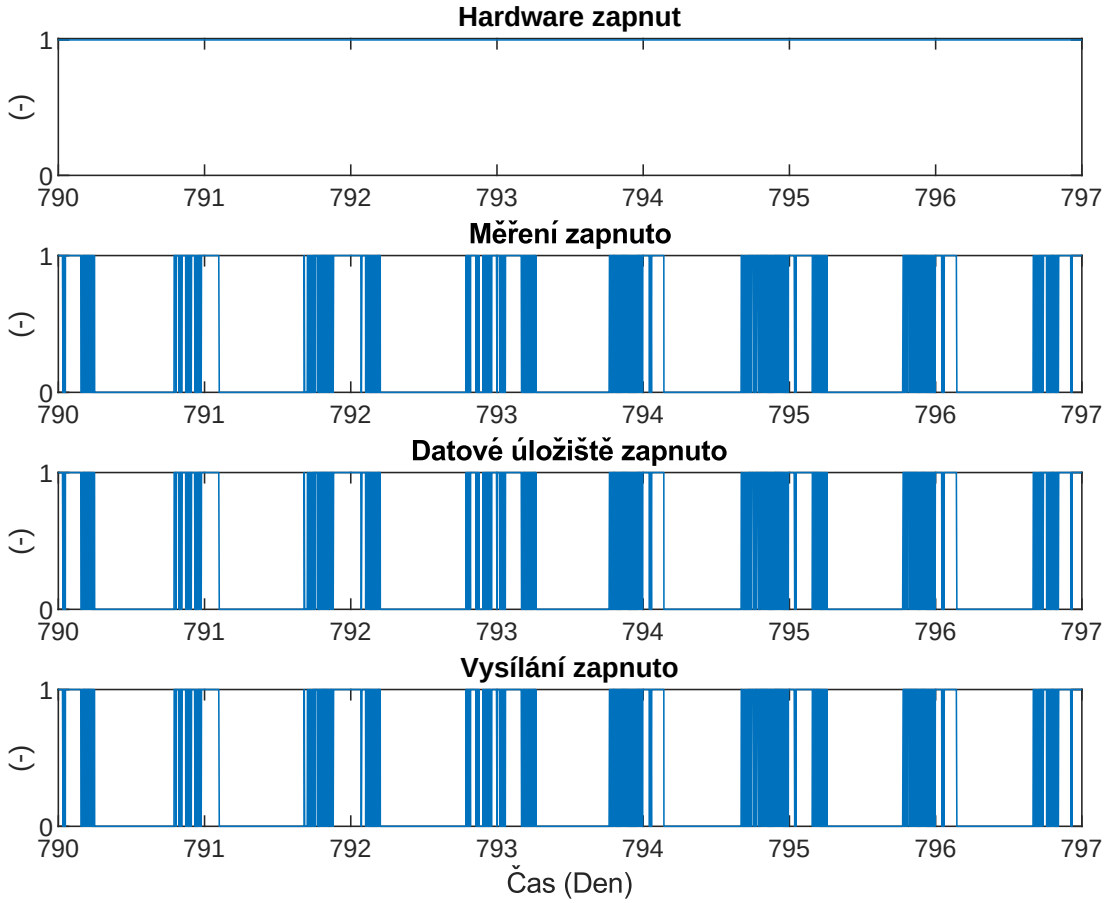
V následujících kapitolách jsou porovnány výsledky algoritmů Q-learning a převzatého časově závislého řízení. Při simulaci byly použity komponenty, které budou realizované na cílovém zařízení. Jedná se hlavně o superkapacitor, který je schopen uložení energie až 750 J a solární panel o ploše 6,48 cm^2 a účinnosti 21 %. Jak již bylo zmíněno v úvodu, simulace byla provedena na vstupních datech slunečního záření od začátku roku 2008 do konce roku 2012.

8.1 Algoritmus Q-learning

Na obrázcích 11 a 12 je vyobrazen průběh energie a ovládání periferií zařízení v průběhu náhodně vybraných sedmi dní. V několika z vybraných dnů došlo k částečnému omezení slunečního záření, pravděpodobně vlivem oblačnosti. To se projevilo na množství vyrobené energie. Vytvořený algoritmus se těmto změnám snadno přizpůsobil. To je patrné z grafu 11 znázorňující zůstatek energie v úložišti po dobu simulace. Ve dnech s nižším slunečním zářením si algoritmus počínal obezřetněji a nereguloval systém až do nízkého stavu, jako v případě dnů s očekávaným množstvím energie.



Obrázek 11: Správa energie algoritmu Q-learning



Obrázek 12: Řízení periferií algoritmu Q-learning

```
16.3.2019 14:41:21 Info Using FirstSimulation
16.3.2019 14:41:22 Info Data check PASS
16.3.2019 14:41:22 Info FairViewDataManager5Min Initialized. Step=60
16.3.2019 14:41:22 Info Using Supercapacitor
16.3.2019 14:41:22 Info Using Solar
16.3.2019 14:41:22 Info Using QLearningController
16.3.2019 14:41:22 Info Using SimpleHardware
16.3.2019 14:41:22 Info Using SimpleDataTransmission
16.3.2019 14:41:22 Info Using SimpleSensing
16.3.2019 14:41:22 Info Using SimpleDataStorage
16.3.2019 14:41:22 Info Using configuration:
HWSNSF.SimulationConfiguration.QLearningConfiguration
16.3.2019 14:41:22 Info Simulation started
16.3.2019 14:43:11 Info Simulation terminated
16.3.2019 14:43:11 Info Simulation OK State CNT: 2538721
16.3.2019 14:43:11 Info Simulation Failure State CNT: 0
16.3.2019 14:43:11 Info Simulation Total steps CNT: 2538721
16.3.2019 14:43:11 Info Supercapacitor Over Charging CNT: 249044
16.3.2019 14:43:11 Info Supercapacitor Over Charging SUM: 800039
16.3.2019 14:43:11 Info Supercapacitor Under Charging SUM: 0
16.3.2019 14:43:11 Info SimpleHardware Enabled CNT: 2538721
16.3.2019 14:43:11 Info SimpleDataTransmission Enabled CNT: 735844
16.3.2019 14:43:11 Info SimpleSensing Enabled CNT: 735844
16.3.2019 14:43:11 Info SimpleDataStorage Enabled CNT: 735844
```

Výstupní parametry simulace z datového logeru jsou uvedeny ve výše vyobrazeném bloku. Při spuštění je vypsáno, jaká simulace byla použita, to lze zadat jako parametr do příkazového řádku. Poté je provedena kontrola vstupních dat a nastaven krok simulace, ten byl v intervalech jedné minuty. Při inicializaci jednotlivých komponent použitých v simulaci jsou zde vypsány. Objekty označeny jako „Simple“ jsou zástupné třídy fyzického zařízení. Je v nich pouze uvedeno, kolik energie daná komponenta spotřebuje za jeden cyklus simulace. Jejich rozšířených funkcí není třeba pro testování řízení systému. V konfiguračním souboru jednotlivých simulací jsou zaregistrovány služby těchto tříd kontejnerem, jak bylo popsáno v kapitole 5.1. Do výstupního souboru je opět zapsána konfigurace simulace, poté je zahájeno zpracovávání dat. Čas simulace je úměrný množství vstupních dat, pro data použita v této práci trvala simulace 8 vteřin. Po ukončení simulace jsou do výstupních souborů zapsány výsledky. Pro adaptivní algoritmus navržený v této práci byly následující. Počet simulačních kroků byl 2 538 721. V žádné z iterací se systém nedostal do stavu selhání, to znamená, že nebyla provedena metoda rollback.

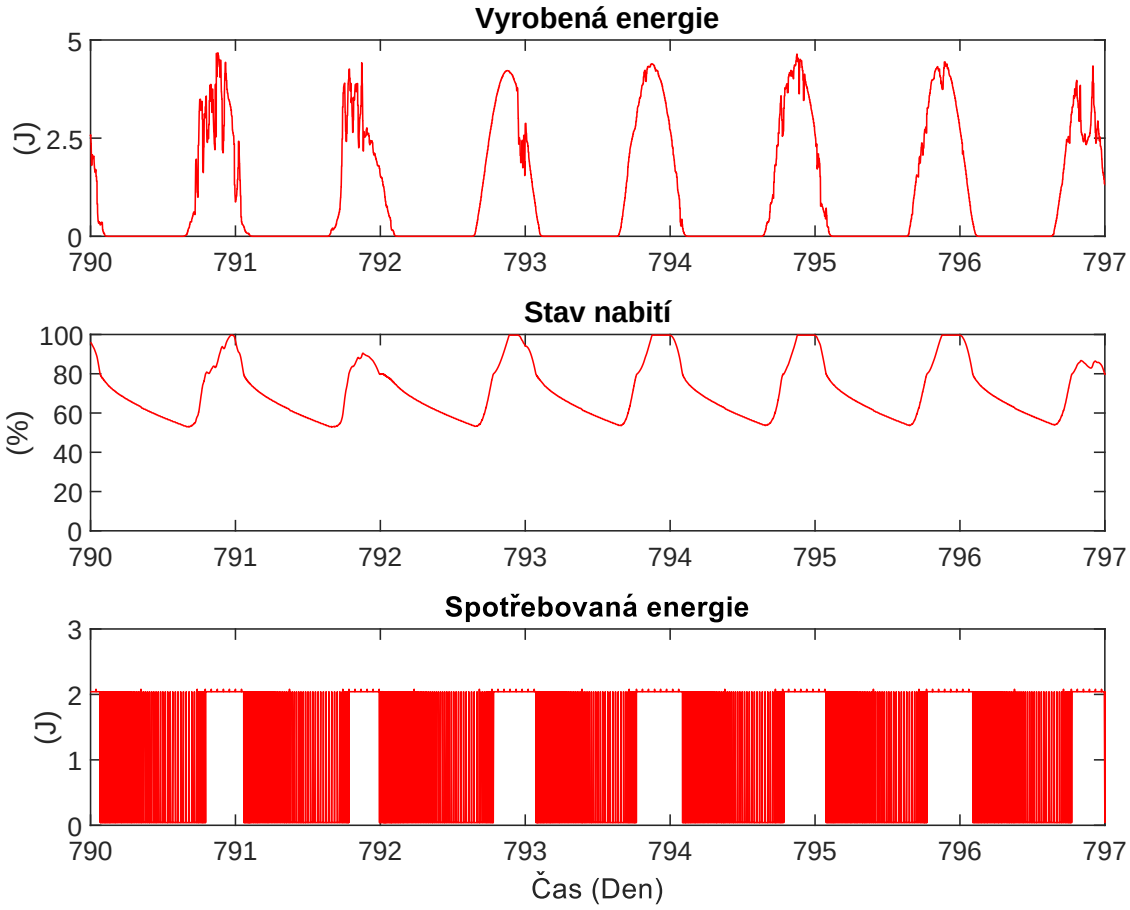
Pro posouzení, zda se algoritmu dařilo efektivně regulovat systém, slouží parametry energetických ztrát. Ty udávají, kolik energie v průběhu simulace kontrolér ztratil v důsledku nabitého energetického úložiště a kolik mu scházelo při jejím nedostatku. Výpadky napájení, kvůli nedostatku energie jsou pro navržený algoritmus nulové, díky tomu také simulace proběhla bez chyb. Výše ztracené energie v důsledku nabitého úložiště dosáhla 800 039 J po dobu 249 044 J iterací, ke ztrátám energie docházelo na konci některých dnů simulace. V momentě, kdy systém dosáhne stavu Full jsou zapnuty všechny periferie a systém se snaží vybít, vydaná energie nemusí být vyšší, než energie vyrobená. To povede k dalším ztrátám. V porovnání s časovým řízením byla ztráta energie v podobě plného energetického úložiště vyšší, než u testovaného algoritmu, avšak počet iterací, ve kterých se systém dostal do plného stavu, je nižší. Rozdílné výsledky u těchto parametrů ukazují na rozdílný počet akcí u obou testovaných algoritmů. Právě díky zvolení pouze dvou akcí v systému se počet hodnot vysílání měření a ukládání dat rovnal. Jejich počet dosáhl 735 844, což odpovídá 3 832 vzorkům během jednoho týdne. Ve chvíli, kdy algoritmus očekával vzrůst energie, začal regulovat systém. Četnost vzorků závisí na očekávaném množství energie v následujícím dnu a množství zůstatku energie z předchozího dne. Oproti tomu hardware systému byl po celý čas simulace zapnutý.

Požadavky na řídicí algoritmus byly splněny, použitý algoritmus neselhal ani jednou v průběhu simulace, a je možno ho otestovat na fyzickém zařízení. Je však nutné připomenout, že řízení nedosahuje optimální strategie, pouze se jí blíží. Tato skutečnost vyplývá z množství ztracené energie. Upravením distribuční funkce lze dále tento parametr minimalizovat. Systém má na výběr ze dvou akcí, zapnutý hardware a zapnuté periferie. Spotřeba energie pro snímání je výrazně vyšší, než u ostatních komponent. Proto bude mít největší vliv na průběh regulace. To dává výhodu časově řízenému kontroléru, který tuto akci zavádí. Může tak začít vybíjet systém dříve a na konci simulačního dne dosáhne nižších energetických ztrát. Zavedení této akce i pro adaptivní algoritmus by mu umožnilo větší flexibilitu a dosažení lepších výsledků. Pro maximalizaci vzorkování byla upravena strategie dle funkce (16). Algoritmus odhaduje optimální řízení, při obdržení nižšího množství energie než je očekáváno není regulace prováděna až do nízkého stavu energie. Je to proto, že v dalším dnu kontrolér očekává také menší množství energie a snaží se přizpůsobit.

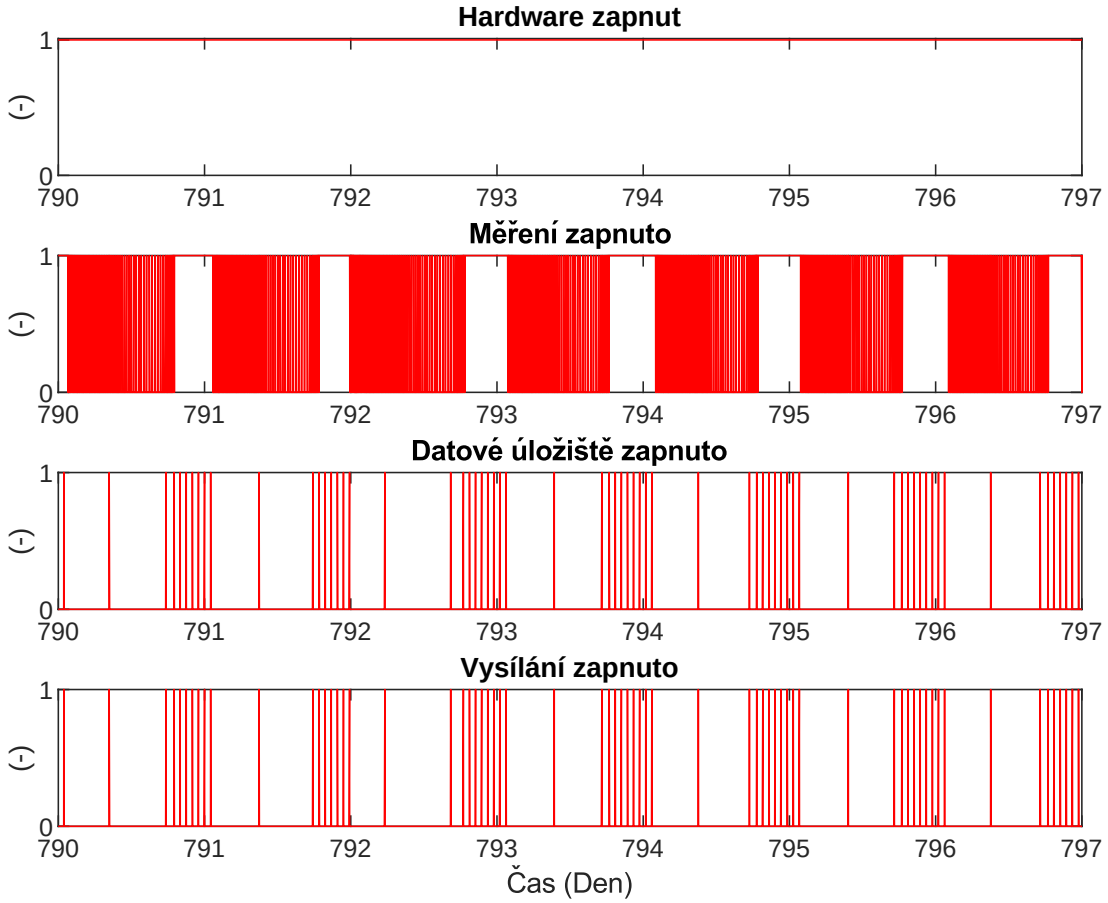
Konstanty učení α a ztráty odměny γ jsou nastaveny relativně vysoko, proto se kontrolér začne rychle adaptovat na nové podmínky a změní způsob regulace, aby dosáhl nastavené strategie. Konstanta průzkumu ϵ byla nastavena rovněž vysoko, jelikož stavový prostor byl relativně malý a nebylo potřeba rozsáhlého průzkumu. Jako možnou nevýhodu tohoto řízení vidím nepravidelné snímání vzorků, i tuto nevýhodu by napravilo zavedení další akce. I přes toto se algoritmus Q-learning vyrovnal ovládání systému časově závislým kontrolérem. Jelikož se oba algoritmy od sebe liší v počtu akcí, je obtížné porovnat, který z nich má navrch. Řízení navrženo v práci má řadu výhod, minimalizuje množství expertních konstant. Schopnost adaptace řízení při umístění zařízení do různých lokalit, ve kterých je různé množství slunečního záření. Naopak nevýhodou je větší složitost algoritmu, z toho vyplývají i vyšší nároky na výpočetní výkon.

8.2 Časově závislé řízení

Algoritmus je převzatý s původního softwaru, jeho chování je porovnáno s adaptivním algoritmem vytvořeným v rámci této práce. Tento přístup řízení spočívá v určení časového úseku, ve kterém se bude provádět daná akce. Časový interval konkrétní akce je dán aktuálním stavem energetického úložiště a v závislosti na předem definované lineární funkci. Zároveň je měřen čas od posledního provedení akce, ten je pak porovnáván s intervalem akce. Pokud je interval přesažen nastaví se proměnné této akce. I tento algoritmus neselhal ani jednou v průběhu simulace a hladina energie se drží nad 50 % energetického úložiště. V algoritmu jsou zpracovány tři druhy akcí, zapnutý hardware, měření a vysílání. I přes jemnější granularitu řízení se tento algoritmus dostal vícekrát do stavu, kdy byl superkategorie plně nabit a docházelo ke ztrátám energie. Časově závislé řízení bylo testováno na stejných vstupních datech jako algoritmus Q-learning. Výhody tohoto algoritmu spočívají v jednoduchosti, zavedením doplňující akce je dosaženo vysokého počtu měření. Vzorkování probíhá v průběhu celého dne. Výpočetní operace nepotřebují velký výpočetní výkon. Výsledky tohoto přístupu řízení v průběhu vybraných sedmi dní jsou vyobrazeny na obrázcích 13 a 14.



Obrázek 13: Správa energie časově závislého kontroléru



Obrázek 14: Řízení periferií časově závislého kontroléru

```
28.11.18 13:53:03 Info Using FirstSimulation
28.11.18 13:53:04 Info Data check PASS
28.11.18 13:53:04 Info FairViewDataManager5Min Initialized. Step=60
28.11.18 13:53:04 Info Using Supercapacitor
28.11.18 13:53:04 Info Using Solar
28.11.18 13:53:04 Info Using TimerBasedController
28.11.18 13:53:04 Info Using SimpleHardware
28.11.18 13:53:04 Info Using SimpleDataTransmission
28.11.18 13:53:04 Info Using SimpleSensing
28.11.18 13:53:04 Info Using SimpleDataStorage
28.11.18 13:53:04 Info Using configuration:
HWSNSF.SimulationConfiguration.FirstSimulationConfiguration
28.11.18 13:53:04 Info Simulation started
28.11.18 14:27:22 Info Simulation terminated
28.11.18 14:27:22 Info Simulation OK State CNT: 2538721
28.11.18 14:27:22 Info Simulation Failure State CNT: 0
28.11.18 14:27:22 Info Simulation Failure Total steps CNT: 2538721
28.11.18 14:27:22 Info Supercapacitor Over Charging CNT: 270498
28.11.18 14:27:22 Info Supercapacitor Over Charging SUM: 612901,152658561
28.11.18 14:27:22 Info Supercapacitor Under Charging SUM: 0
28.11.18 14:27:22 Info SimpleHardware Enabled CNT: 2538721
28.11.18 14:27:22 Info SimpleDataTransmission Enabled CNT: 16320
28.11.18 14:27:22 Info SimpleSensing Enabled CNT: 824003
28.11.18 14:27:22 Info SimpleDataStorage Enabled CNT: 16320
```

Výpis výše znázorňuje výsledky za celou dobu simulace. Sbírání vzorků probíhalo téměř v polovině iterací a na konci simulace bylo pořízeno 824 003 vzorků. Počet vzorků je o 88 159 vyšší, než jak tomu bylo u adaptivního algoritmu. To je důsledkem přidání třetí akce, což zvýšilo prioritu u parametru snímání. Zvýšení počtu vzorků však výrazně snížilo počet vysílání a ukládání do paměti. Během průběhu simulace bylo datové úložiště a vysílání zapnuto ve 16 320 případech. Podle množství energie při plném nabití superkapacitoru lze usoudit, že tento přístup byl efektivnější, než předešlý. Na konci simulace byla tato energie rovna 612 901 J. Do stavu, kdy bylo energetické úložiště nabité, se však dostal vícekrát než adaptivní algoritmus, jak bylo zmíněno v předešlé kapitole.

Nevýhody tohoto algoritmu vyplývají z neschopnosti adaptace na změnu podmínek. Zároveň je potřeba experimentálního nastavení křivek řízení tak, aby nedocházelo k nedostatku energie. Výsledky obou algoritmů jsou v simulačním prostředí srovnatelné, teprve testování na reálném zařízení dokáže odhalit nedostatky, které se v simulaci nemusely projevit.

8.3 Testování vybraného algoritmu na bludišti

Pro otestování algoritmu Q-learning a jeho schopností jsem vytvořil 2D bludiště, ve kterém bude algoritmus implementován. Jde o klasický typ úlohy nalezení nejkratší cesty, dle toho byla také nastavena strategie agenta.

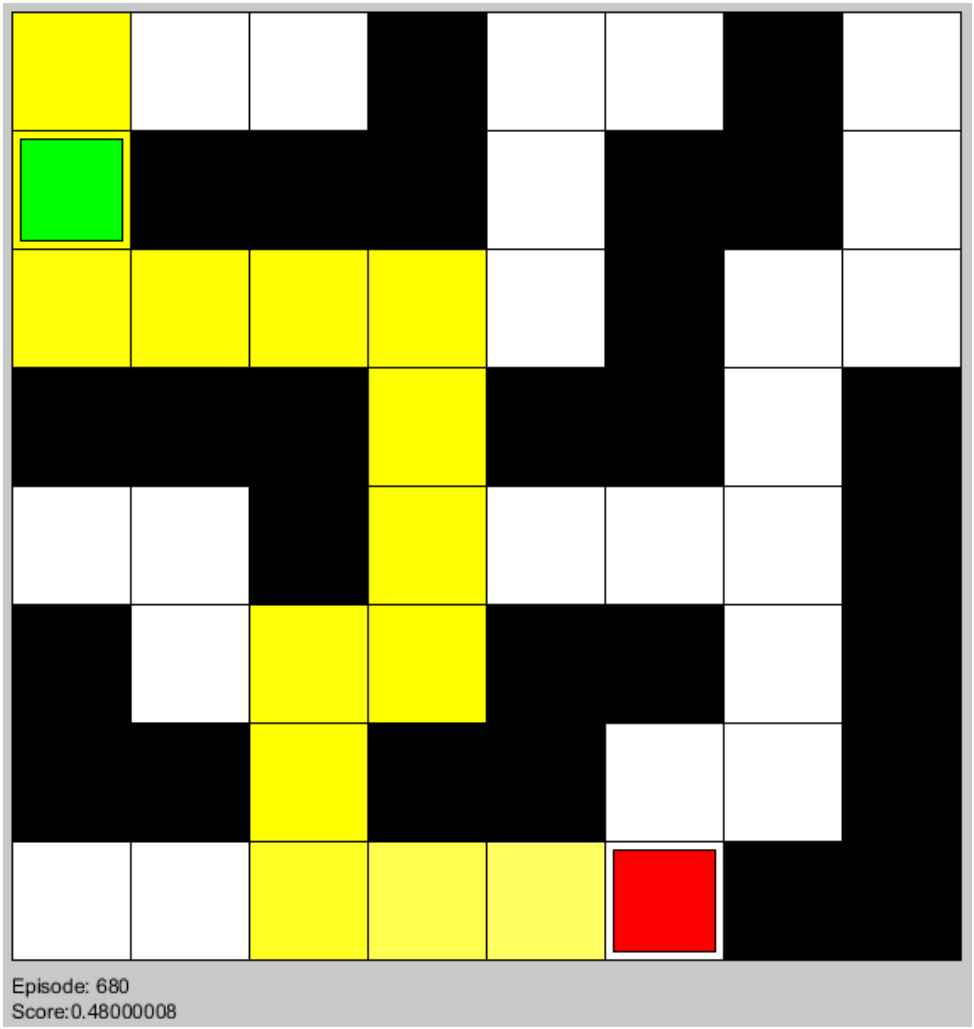
$$R(S,A) = \begin{cases} -1 & S \text{ je černé pole} \\ 1 & S \text{ je červené pole} \\ -0,04 & \text{Za každý krok} \end{cases} \tag{17}$$

Při vstupu agenta na červené pole obdrží odměnu 1, epizoda je ukončena a agent se vrací do startovacího pole (levý horní roh). Vkročení do černého pole je udělena záporná odměna -1, i zde je provedeno ukončení epizody. S každým krokem, který agent zvolí mu je udělena odměna -0,04 a to z důvodu nalezení nejkratší cesty. Tato strategie řízení je dána rovnicí (17). Agent se tedy snaží minimalizovat ztráty. Při odstranění tohoto bodu strategie bude vše fungovat, ale v určitém procentu případů se agent naučí delší trasu. Učení bludiště s větším počtem překážek trvá podstatně déle, než při zvolení jednoduchého bludiště. Pro případ na obrázku 15 to bylo zhruba 500 epizod. Rychlost učení je úměrná složitosti bludiště a velikosti stavového prostoru, jde snížit dynamickým měněním expertních konstant. Pro názorný příklad byly konstanty ponechány staticky a to $\epsilon = 0,9$, $\alpha = 0,1$ a $\gamma = 0,9$. Vizualizační prostředí je napsáno formou třídy, zacházení s ní je tedy velice jednoduché. A k funkčnosti algoritmu stačí pár příkazů v hlavní smyčce.

```
1    void draw(){
2       background(200);
3       //vykreslení prostředí
4       env.Draw(player);
5       //výběr nejlepší akce
6       int action = ChooseAction(observation);
7       //provedení iterace prostředí
8       Container data = env.Step(action);
9       //aktualizace Q-tabulky
10      learn(observation, data.state_, data.reward, action);
11      //posunutí agenta do nového stavu
12      observation[0] = player[0];
13      observation[1] = player[1];
14      if (data.done)
15         env.Reset(); //reset epizody
16   }
```

Výpis 6: Hlavní smyčka algoritmu pro vyhledání cesty v bludišti

Metodou background se provede vymazání prostředí, poté je opětovně vykresleno s aktualizovanou pozicí agenta. Posun agenta a odměnu zajistí funkce step. Následně je provedena aktualizace hodnot Q-tabulky funkcí learn. V neposlední řadě je potřeba agenta posunout na novou pozici. Takto je cyklus opakován do doby ukončení epizody. V případě ukončení epizody je prostředím vrácen příznak ukončení, což vede k resetování pozice agenta. Pro vizualizaci celé úlohy jsem zvolil Processing 3.4, jelikož byla aplikace cílená na Windows. Processing je založen na programovacím jazyku Java a nabízí celou řadu vizualizačních nástrojů, proto byl pro tuto úlohu ideální. Aplikace jsem také napsal v Pythonu3.0 pro použití na Linuxu, zde je ovšem potřeba instalace knihovny Tkinker pro vizualizaci.



Obrázek 15: Aplikace Q-learning ve 2D bludišti

9 Závěr

Cílem této diplomové práce bylo vytvořit adaptivní řídicí algoritmus, který je schopen optimálně využít dostupnou energii v nezávislém vestavěném systému. V první části práce jsou popsány principy, výhody a možnosti energeticky nezávislého vestavěného systému. Další kapitoly se zabývají strojovým učením, zejména pak zpětnovazebním učením. Zde jsou vysvětleny nejpoužívanější algoritmy této oblasti, jejich možnosti a použití. Pro algoritmus Q-learning, který byl použit pro řízení simulačního prostředí, zde byl uveden praktický příklad pro demonstraci jeho účinnosti. Jde o přístup nalezení nejkratší cesty v neznámém prostředí. Práce se opírá o předchozí publikace zpracovávající totožný problém a rozšiřuje tak možnosti řešení této problematiky.

V rámci diplomové práce byl vytvořen software, který matematicky popisuje chování použitých komponentů na fyzickém zařízení. To umožnilo testování jednotlivých algoritmů. Část simulačního prostředí tvoří logovací systém, který byl vpraven do ostatních komponent. Skládá se z chybového a datového logeru. Datový loger efektivně skládá výstupní data simulace a chybový poskytuje informace o stavu simulace a případných chybách. Logovací systém byl víceúrovňový, přetěžování metod odstranilo nutnost přetypování, což usnadnilo použití.

V práci byly porovnány algoritmy časového a adaptivního řízení. Časově závislé řízení bylo převzato. Oba algoritmy prokázaly schopnost dlouhodobého řízení systému bez výpadků energie. Algoritmus Q-learning se dokázal úspěšně adaptovat v situacích, kdy systém obdržel nižší množství energie. Základní princip strategie kontroléru spočívá v kladném bodování agenta, kdy systém držel ve středním pásmu energie elektrického úložiště a záporném v případě překročení tohoto pásma. Strategie je doplněna o pravidla minimalizující výpadky energie a energetické ztráty při plně nabitém energetickém úložišti. Pomocí lineárních funkcí závislých na slunečním cyklu bylo energetické úložiště rozděleno do stavů, se kterými algoritmus pracoval. Expertní konstanty byly zvoleny staticky a díky malému stavovému prostoru došlo k rychlému prozkoumání všech možných stavů systému. Implementovaný algoritmus neselhal ani jednou během simulačního času. Energetické úložiště bylo v počátku simulace nabito, agent tak měl dost času naučit se systém ovládat. Konstanta průzkumu byla nastavena vysoko, což zamezilo volbě nevhodné akce. Simulace byla prováděna na datech slunečního záření v průběhu čtyř let s časovým rozestupem vzorků pět minut.

V simulačním prostředí se nacházely všechny komponenty energeticky nezávislého vestavěného systému, přičemž bloky byly implementovány jako třídy. Při potřebě přístupu na metody nebo proměnné z jedné komponenty uvnitř druhé by mezi nimi vznikla nežádoucí vazba, která by omezovala modulárnost systému a umožňovala rozsáhlejší šíření chyb. Z toho důvodu byl použit moderní návrhový vzor dependency injection, který této vazbě zabraňuje. Do systému je tak možno vkládat třídy s různými parametry fyzických zařízení, bez nutnosti přepisu předchozích.

Algoritmus popsaný v práci umožňuje optimální využití omezeného množství energie a bude použit na fyzickém zařízení, které bude monitorovat ekosystém. Metody zpětnovazebního řízení mají v oblasti nezávislých systémů vysoký potenciál využití, jelikož snižují cenu potřebnou pro

údržbu systému. V oblasti měření environmentálních veličin je důležité, aby hodnoty nebyly narušeny přítomností člověka, proto je zde klíčové použít energeticky nezávislý systém. Rozvoj práce spočívá v aplikaci navrženého algoritmu na fyzické zařízení a otestování chování v reálném světě. Lokalita umístěného zařízení bude mít značný vliv na jeho řízení. Chování systému se blíží optimálnímu řízení, strategii lze dále vylepšovat a tím dále minimalizovat energetické ztráty. Použití jiného algoritmu zpětnovazebního učení by rovněž mohlo vést ke zlepšení chování systému.

Literatura

[1] Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 176–185. JMLR. org, 2017. (Citováno dne: 13. 1. 2019).

[2] David Čapka. *Softwarové architektury a dependency injection*, leden 2018. (Citováno dne: 27. 12. 2018), Dostupné z:https://www.itnetwork.cz/navrh/architektury-a-dependency-injection.

[3] Mustafa Engin Basoglu and Bekir Cakir. Comparisons of mppt performances of isolated and non-isolated dc–dc converters by using a new approach. *Renewable and Sustainable Energy Reviews*, 60:1100–1113, 2016. (Citováno dne: 16. 2. 2019).

[4] Bonsai. *Deep Reinforcement Learning Models: Tips and Tricks for Writing Reward Functions*, listopad 2017. (Citováno dne: 6. 1. 2019), Dostupné z:https://medium.com/@BonsaiAI/deep-reinforcement-learning-models-tips-tricks-for-writing-reward-functions-a84fe525e8e0.

[5] Denny Britz. *Exploration vs. Exploitation*, duben 2014. (Citováno dne: 19. 10. 2018), Dostupné z:https://medium.com/@dennybritz/exploration-vs-exploitation-f46af4cf62fe.

[6] Jason Brownlee. *Supervised and Unsupervised Machine Learning Algorithms*, březen 2016. (Citováno dne: 07. 11. 2018), Dostupné z:https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/.

[7] Bruce Dunn, Haresh Kamath, and Jean-Marie Tarascon. Electrical energy storage for the grid: a battery of choices. *Science*, 334(6058):928–935, 2011. (Citováno dne: 24. 11. 2018).

[8] Peter Fielder, PG Comeau, et al. *Construction and testing of an inexpensive PAR sensor*, volume 53. Citeseer, 2000. (Citováno dne: 26. 2. 2019).

[9] Katerina Fragkiadaki. *Markov Decision Processes*, březen 2017. (Citováno dne: 13. 10. 2018), Dostupné z:https://www.cs.cmu.edu/~katef/DeepRLControlCourse/lectures/lecture2_mdps.pdf.

[10] Karan M Gupta. Performance comparison of sarsa (λ) and watkin's q (λ) algorithms. *nd): n. pag. Print.* (Citováno dne: 11. 11. 2018).

[11] Jane Hodgkinson, Richard Smith, Wah On Ho, John R Saffell, and Ralph P Tatam. Non-dispersive infra-red (ndir) measurement of carbon dioxide at 4.2 μm in a compact and optically efficient sensor. *Sensors and Actuators B: Chemical*, 186:580–588, 2013. (Citováno dne: 24. 2. 2019).

[12] Ronald A Howard. Dynamic programming and markov processes. 1960. (Citováno dne: 8. 11. 2018).

[13] Jason Hsu, Sadaf Zahedi, Aman Kansal, Mani Srivastava, and Vijay Raghunathan. Adaptive duty cycling for energy harvesting systems. In *Proceedings of the 2006 international symposium on Low power electronics and design*, pages 180–185. ACM, 2006. (Citováno dne: 22. 2. 2019).

[14] Steeve Huang. *Introduction to Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG)*, leden 2017. (Citováno dne: 18. 10. 2018), Dostupné z:https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287.

[15] Apogee Instruments. *Model SQ-500 - Full-spectrum Quantum Sensor*, 2016. (Citováno dne: 25. 02. 2019), Dostupné z:https://www.agriculture-xprt.com/products/apogee-model-sq-500-full-spectrum-quantum-sensor-541676.

[16] Arthur Jiliani. *Simple Reinforcement Learning with Tensorflow Part 7: Action-Selection Strategies for Exploration*, listopad 2016. (Citováno dne: 28. 10. 2018), Dostupné z:https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-7-action-selection-strategies-for-exploration-d3a97b7cceaf.

[17] Vladimír Kašík. Programování hradlových polí. pages 18–19, 2012. (Citováno dne: 20. 12. 2018).

[18] Raj Kamal. *Embedded systems: architecture, programming and design*. Tata McGraw-Hill Education, 2011. (Citováno dne: 24. 2. 2019).

[19] Alireza Khaligh and Omer C Onar. *Energy harvesting: solar, wind, and ocean energy conversion systems*. CRC press, 2009. (Citováno dne: 1. 3. 2019).

[20] Tarik Kousksou, Pascal Bruel, Abdelmajid Jamil, T El Rhafiki, and Youssef Zeraouli. Energy storage: Applications and challenges. *Solar Energy Materials and Solar Cells*, 120:59–80, 2014. (Citováno dne: 6. 12. 2018).

[21] Trong Nhan Le, Alain Pegatoquet, Olivier Berder, Olivier Sentieys, and Arnaud Carer. Energy-neutral design framework for supercapacitor-based autonomous wireless sensor networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 12(2):19, 2015. (Citováno dne: 22. 2. 2019).

[22] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM, listopad 2016. (Citováno dne: 6. 11. 2018).

[23] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, prosinec 2013. (Citováno dne: 18. 10. 2018).

[24] Codrut Negau. *DIGITAL CITIZEN, System on a chip*, květen 2017. (Citováno dne: 17. 2. 2019), Dostupné z:https://www.digitalcitizen.life/soc-system-on-chip.

[25] Luca Palmieri. *Reinforcement Learning: a comprehensive introduction*, květen 2018. (Citováno dne: 28. 10. 2018), Dostupné z:https://www.lpalmieri.com/posts/rl-introduction-00/.

[26] Xue Bin Peng, Angjoo Kanazawa, Jitendra Malik, Pieter Abbeel, and Sergey Levine. Sfv: Reinforcement learning of physical skills from videos. *ACM Trans. Graph.*, 37(6), November 2018. (Citováno dne: 24. 11. 2018).

[27] Michal Prauzek. *ČÍSLICOVÁ A MIKROPROCESOROVÁ TECHNIKA*, 2013. (Citováno dne: 11. 11. 2018), Dostupné z:https://homel.vsb.cz/~pra132/files/CMT_prauzek_final_1_2.pdf.

[28] Michal Prauzek. Metody řízení pro energeticky nezávislé vestavěné měřící systémy. 2017. (Citováno dne: 16. 2. 2019).

[29] Michal Prauzek, Jaromir Konecny, Monika Borova, Karolina Janosova, Jakub Hlavica, and Petr Musilek. Energy harvesting sources, storage devices and system topologies for environmental wireless sensor networks: A review. *Sensors*, 18(8):2446, 2018. (Citováno dne: 7. 12. 2018).

[30] Vijay Raghunathan, Aman Kansal, Jason Hsu, Jonathan Friedman, and Mani Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 64. IEEE Press, 2005. (Citováno dne: 2. 12. 2018).

[31] Steve Roberts. *DC/DC book of knowledge: Practical tips for the User*. Recom, 2015. (Citováno dne: 23. 2. 2019).

[32] Shad Roundy, Dan Steingart, Luc Frechette, Paul Wright, and Jan Rabaey. Power sources for wireless sensor networks. In *European workshop on wireless sensor networks*, pages 1–17. Springer, 2004. (Citováno dne: 9. 12. 2018).

[33] Shad Roundy, Paul K Wright, and Jan Rabaey. A study of low level vibrations as a power source for wireless sensor nodes. *Computer communications*, 26(11):1131–1144, 2003. (Citováno dne: 8. 12. 2018).

[34] Najib Akraml Ruicong Xie. *Markov Decision Processes (MDPs) - Structuring a Reinforcement Learning Problem*, září 2018. (Citováno dne: 01. 11. 2018), Dostupné z:http://deeplizard.com/learn/video/my207WNoeyA.

[35] Harm Seijen and Rich Sutton. True online td (lambda). In *International Conference on Machine Learning*, pages 692–700, 2014. (Citováno dne: 16. 12. 2018).

[36] Andrew Sendy. Pros and cons of monocrystalline vs polycrystalline solar panels. *Retrieved December*, 16, 2016. (Citováno dne: 16. 2. 2019).

[37] Henry A Sodano, Daniel J Inman, and Gyuhae Park. A review of power harvesting from vibration using piezoelectric materials. *Shock and Vibration Digest*, 36(3):197–206, 2004. (Citováno dne: 14. 12. 2018).

[38] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, listopad 2018. (Citováno dne: 05. 10. 2018).

[39] Anthony Knittel Tim Eden, Raphael van Uffelen. *Reinforcement Learning: TD-Learning*, květen 2015. (Citováno dne: 29. 10. 2018), Dostupné z:https://www.cse.unsw.edu.au/~cs9417ml/RL1/tdlearning.html.

[40] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992. (Citováno dne: 28. 11. 2018).

Přílohy

Příloha A Skript na zpracování dat Příloha v IS EDISON

Příloha B Testování RL na bludišti Příloha v IS EDISON